REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	18 Apr 97	

4. TITLE AND SUBTITLE
LEARNING HIDDEN STRUCTURE FROM DATA: A METHOD FOR MARGINALIZING JOINT DISTRIBUTIONS USING MINIMUM CROSS-CORRELATION ERROR

5. FUNDING NUMBERS

6. AUTHOR(S)
ANTHONY KAHLIL HAYNES

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

8. PERFORMING ORGANIZATION REPORT NUMBER
97-020

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
DEPARTMENT OF THE AIR FORCE
AFIT/CI
2950 P STREET
WRIGHT-PATTERSON AFB OH 45433-7765

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

19970424 030

14. SUBJECT TERMS	15. NUMBER OF PAGES
	72
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

LEARNING HIDDEN STRUCTURE FROM DATA: A METHOD FOR MARGINALIZING JOINT DISTRIBUTIONS USING MINIMUM CROSS-CORRELATION ERROR

BY

ANTONY KAHLIL HAYNES

B.S., United States Air Force Academy, 1995

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1997

Urbana, Illinois

LEARNING HIDDEN STRUCTURE FROM DATA

Antony Kahlil Haynes, M.S.

Department of Computer Science

University of Illinois at Urbana-Champaign, 1997

David C. Wilkins, Advisor; Sylvian Ray, Co-Advisor

This thesis demonstrates an entropy-based, Bayesian dependency algorithm—*Minimum Error Tree Decomposition II* (METD2)—that performs computer-based generation of probabilistic hidden-structure domain models from a database of cases. The system learns probabilistic hidden-structure domain models from data, which partially automates the task of expert system construction and the task of scientific discovery. Existing probabilistic systems find associations among the observable variables but do not consider the presence of hidden variables, or else, do not use cross-correlation error as the metric for building the hidden structure. The algorithm decomposes a joint distribution of n observable variables into $n+1$ observable and hidden variables. The hidden variable exists in the form of a tree consisting of $n-1$ interior nodes. The final product of the procedure is a combined tree whose n leaves are the observable variables in a sample and whose $n-1$ interior nodes are the marginalizations for the leaves.

ACKNOWLEDGMENTS

I would, first and foremost, like to thank my research advisor, Dr. David C. Wilkins, for proposing the idea for this research and providing computing facilities, technical support, and funds for me to complete it. Without his sponsorship this research would not have been done.

Next, I would like to thank my co-advisor, Dr. Sylvian Ray, for providing guidance in the related area or artificial neural networks, whose similarity to Bayesian belief networks and computational advantages made their study essential to the completion of this thesis.

I would like to thank the U. S. Air Force Academy and the Air Force for awarding me a graduate scholarship, and allowing me to attend graduate school and pursue advance studies.

Also, discussions with members of the Knowledge-Based Systems Group benefited this thesis enormously. In particular, I would like to thank Bill Hsu and Ole Mengshoel for their help in understanding prior research in this area. In addition, the environment of intellectual curiosity sponsored by the regulars of AI West, also provided constant intellectual and morale support. I wish them the best of luck in their research and professional endeavors.

In addition, I would like to thank my parents and brother for their guidance, love, and support. Without them I would not be here to publish this work. I am always indebted to them.

Finally, I would like to thank our Lord and Savior, Jesus Christ, without whom, none of us would be here.

TABLE OF CONTENTS

1. INTRODUCTION

In this thesis, I demonstrate an entropy-based, Bayesian dependency algorithm that performs computer-based generation of probabilistic hidden-structure domain models from a database of cases. A case is simply a randomly-drawn sample of observations about the states of some domain or system, e.g. the vital statistics for a patient at some point in time, the market indicators for some financial system, or the results of diagnostic test applied to an automobile. Existing approaches to generating computer-based domain models do not explicitly reconstruct hidden structure, or else, assume a minimal, one-element, hidden structure for the entire domain model [Cheesman et al, 1989], [Heckerman, 1995], [Herskovits, 1991]. In the past decade, research and development of Bayesian belief network systems encoding human expertise have greatly increased [Jensen, 1996]. Jensen lists several such applications: BOBLO (agriculture), Computer Vision, Mildew (Agriculture), Wizard (Microsoft), VISTA (NASA), MUNIN (Medicine), Hailfinder (Weather), FRAIL (prose writing), and the Army Battlefield Reasoner (Military). At the same time, the government, corporations, and academia have accumulated large domain databases, such as life-insurance actuarial tables, census statistics, medical data, and astronomical spectral repositories. Applications of the probabilistic paradigm to these new domains would benefit from a way of learning structure from data.

Such a system of learning a probabilistic hidden-structure domain model from data helps automate the task of construction an expert system, which may then be applied to make decisions, classify new cases, or predict a future system state. A second application is automated scientific discovery, where the Bayesian network system constructs hidden structures (theories) from the cases contained in some sample database. Here, the expert system derives knowledge from domains for which expertise is not available or is poorly developed. A third application is to use a belief net to perform automated hypothesis testing, where a researcher makes a guess that some arbitrary relationship between variables holds and the Bayesian network returns the likelihood of such a relationship.

Bayesian network systems address two general problems central to the field of artificial intelligence: reasoning under uncertainty and learning. Bayesian belief networks differ from other approaches to learning in that they have a firm mathematical foundation in probability theory. In addition, the technology of belief networks is highly expressive and easy to apply. For example, AutoClass III is an automated Bayesian-probabilistic classifying system which conditions statistically-based hypothesis on a single hidden variable [Cheesman et al, 1989]. Also, the Microsoft Answer Wizard is a well-known commercial expert system implemented using a Bayesian belief network. Existing approaches to finding hidden structure either do not assume a tree-hidden structure or require more complex algorithmic analysis. For example, Kutato [Herskovits, 1988] finds associations between observable variables, but does not explore possible hidden variable relationships.

I have chosen in this thesis to work with models that are

- tree-structured (directed-acyclic graphs, with arcs representing associations among the hidden variables (interior nodes) and observable variables (leaf nodes))
- non-parametric (the hidden variables are added to the tree structure without reference to a particular distribution)
- probabilistic (the model's parameters determine a joint probability distribution over the hidden and observable variables).

Existing probabilistic systems find associations among the observable variables but do not consider the presence of hidden variables, or else, do not use cross-correlation error as the metric for building the hidden structure. Also, many algorithms for finding structure do not have a normative halting criteria and have lengthy running times, as noted in [Herskovits, 1991]. Given the explosive growth in large databases in all fields of life, researchers, businessmen, and government agencies all may benefit from computationally efficient, non-parametric, domain-independent systems which generate probabilistic hidden-structure domain models from databases. In this thesis, I present one such algorithm, Minimum Error Tree Decomposition, the earliest version of this approach is found in [Liu, 1990].

1.1 Definitions and Background

The conceptual foundation for this research lies in statistics, probability theory, and artificial intelligence. Thus, I first describe probability and correlation, as defined by mathematicians and as it relates to this research. Next, I provide a precise description of what is a Bayesian belief network and what is a tree-structured network. Last, I define the problem of finding a Bayesian network hidden structure given a database of observable variables.

1.1.1 Probability and Correlation

We may view probability as the likelihood of some event occurring. For example, on a given day we choose to carry an umbrella or not based on our estimation of the likelihood of it raining on that day; if we do take the umbrella, will we forget it; and if we forget it, will it be stolen. The probabilities ***P(•)*** might be represented as

$$P(\textit{rain forecasted}) = 0.3$$
$$P(\textit{umbrella taken} \mid \textit{rain forecasted}) = 0.9$$
$$P(\textit{umbrella forgotten} \mid \textit{umbrella taken}) = 0.1$$
$$P(\textit{umbrella stolen} \mid \textit{umbrella forgotten}) = 0.01.$$

Figure 1.1 Conditional Probabilities for Umbrella Scenario.

With the probabilities defined in Figure 1.1, we can determine the likelihood of our umbrella being stolen on any given day by "chaining" the conditional probabilities:

$$P(\textit{umbrella stolen}) = P(\textit{umbrella stolen} \mid \textit{umbrella forgotten})\, P(\textit{umbrella forgotten} \mid \textit{umbrella taken})\, P(\textit{umbrella taken} \mid \textit{rain forecasted})\, P(\textit{rain forecasted})$$

$$P(\textit{umbrella stolen}) = 0.01 \text{ X } 0.1 \text{ X } 0.9 \text{ X } 0.3 = 0.00027 \text{ or less than } 0.3\ \%.$$

The probability that an event will occur, in the absence of any other knowledge about the world may be defined as the *prior* probability, since it the probability of an event prior to any other knowledge. In the umbrella scenario, each of the conditional probabilities listed in Figure 1.1. are prior probabilities. On the other hand, *posterior* probabilities take into account knowledge that we have gained about the world, and thus are the likelihood of an event given some knowledge. The probability that our umbrella is stolen in the umbrella scenario is the only posterior probability in the previous example. The general formula for relating the posterior and prior probabilities of two variables, ***x*** and ***y***, in terms of some evidence context, ***e*** is

$$P(x|y) = \frac{P(y|x)}{P(y|e)} P(x|e), \quad P(y|e) > 0 \tag{1}$$

The prior probability is denoted $P(x \mid e)$, and the posterior probability is denoted $P(x \mid y, e)$. Normally, we neglect the evidence context, e, (as we did in the Umbrella Scenario of Figure 1.1) since it is usually clear from the problem. [Pearl, 1988], [Neopolitan, 1990], and [Heckerman, 1995] explain in some detail the mechanisms of Bayesian probability.

The correlation coefficient is a number between -1 and +1 which helps determine how closely related are two variables. If two variables are completely dependent upon each other, then the measure is 0, if they are completely independent, its absolute value is 1. The correlation ρ_{xy} between two random statistical variables X and Y is defined as

$$\rho_{XY} = \frac{E(XY) - E(X)E(Y)}{\sqrt{Var(X)}\ \sqrt{Var(Y)}}, \tag{2}$$

where $E(X)$ is the expected (mean) value of the random variable X and $Var(X)$ is the variance. In the case of binary random variables the expectation and variance of each variable, X, is

$$E(X) = P(X), \qquad Var(X) = 1 - E(X)$$

which yields the correlation coefficient as

$$\rho_{XY} = \frac{P(XY) - P(X)P(Y)}{\sqrt{P(X)[1-P(X)]}\ \sqrt{P(Y)[1-P(Y)]}}.$$

1.1.2 Belief Networks

Belief networks are graphical representation methods which encode an expert's knowledge about a specified problem domain. The nodes in the graph represent specific states—variables,

hypothesis, or tests—in the domain. The directed arcs between nodes represent probabilistic dependencies which are valid between graph nodes. Direction may be interpreted as causation. Given a probabilistic network, the probability of the network is given by the chain rule:

$$P(x_1, \cdots, x_n|e) = \prod_{i=1}^{n} P(x_i|x_1, \cdots, x_{i-1}, e) \tag{3}$$

We informally applied the chain rule in the umbrella scenario from Figure 1.1 to get the posterior probability of the umbrella being stolen.

More formally, a Bayesian belief network is a graphical representation method in which nodes represent domain variables and arcs between nodes represent probabilistic dependencies [Cooper and Herskovits, 1992]. A belief network $\boldsymbol{B} = (\boldsymbol{B_s}, \boldsymbol{B_p})$ where $\boldsymbol{B_s}$ is a directed-acyclic graph (DAG) and $\boldsymbol{B_p}$ is a Markov probability field.

The problem of finding hidden structure is important because it can be used to simplify the complexity of belief network architectures, in addition to performing hypothesis testing and automated scientific discovery. For example, say that we have five different people telling each other the time. One possible interpretation of their observations, is that the time reported by each person depends upon the time reported by all others, as shown in Figure 1.2 (a). A second interpretation, is that there is one *hidden variable*, the true time, and that each person's observation depends upon this hidden variable, as shown in Figure 1.2 (b). Thus, instead of having $n!$ relationships to consider, the new model only has n relationships. The hidden node abstracts the idea that there is one true time, which makes the problem conceptually cleaner. Likewise, in large databases, assuming the existence of hidden variables simplifies the complexities of the resulting Bayesian belief domain model constructed from the data.

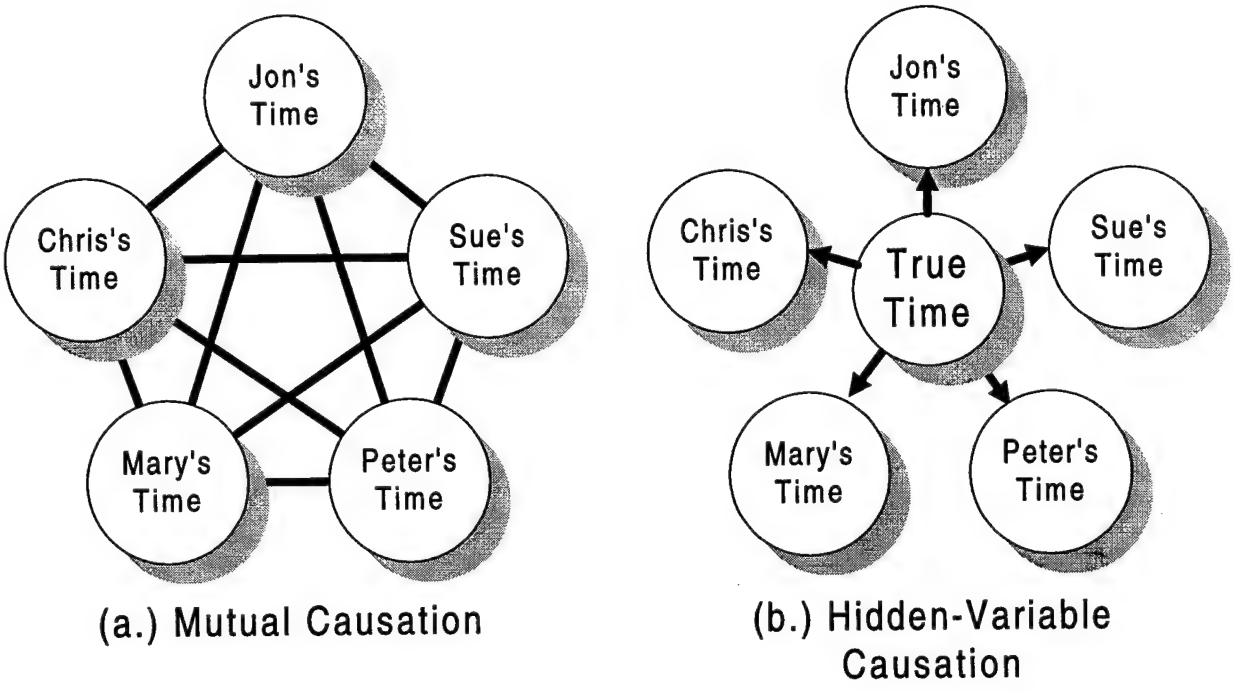


Figure 1.2 Hidden Structure in Telling the Time.

The network in Figure 1.2 (b) reduces the complexity of the telling the time problem by mediating the possible dependencies between observable variables (each person's time) with a central variable (the true time). This mediation between variables makes (b) a *modular architecture* [Pearl, 1988; 383]. Thus (b) is a more effective data structure than (a) for representing the empirical knowledge of telling the time because it can be queried and updated at a higher speed (since it has fewer connections and a common, centralized variable). That is, given knowledge of the true time (the central variable) all the (observable) variables are independent of each other. The class of structures represented by both (a) and (b) of Figure 1.2 are *causal structures* because they conceptually match empirical phenomena to empirical data,

and the central variable in (b) is a *causal variable* because it permits us to treat the observable variables as independent once it is calculated [Pearl, 1988; 383].

At last, we come to the rationale for using tree-structured architectures. From our analysis of Figure 1.2, we see that we would like architectures with the characteristics of (b)—that are centrally coordinated. Both star-structured and tree-structured networks have this property. As noted by [Pearl, 1988; 386], tree structures have several advantages for finding hidden structures:

- Every pair of non-adjacent variables are independent given a third variable on the path connecting the pair.
- A node is certain that information provided by each neighbor originates from an independent source (allows application of local combination rule).
- Conversion/reconfiguration of other structures to trees has already been pioneered in [Chow and Liu, 1968]—If the leaf variable for some tree T with hidden values cannot be separated and would lead to a complete graph, the by adding hidden "dummy variables" we can construct a tree and thereby gain its advantage.
- Trees are compact—leads to storage economy.
- Required parameters $P(x_i|x_j)$ can be estimated reliably from data.
- Tree have a computational advantage in data interpretation.

Thus, equation (3), the general chain rule for belief models, simplifies, under the assumptions listed above for a tree-decomposable data, to a tree-dependent distribution, $P^t(x)$,

$$P^t(\bar{x}) = \prod_{i=1}^{n} P\left(x_i \middle| pa(x_i)\right) \tag{4}$$

where P^t is a Markov field relative to the tree t, and $pa(x_i)$ is the variable designated as the parent of x_i.

Thus, if a belief network (B_s, B_p) is a tree-structured model with a tree-dependent probability distribution, then $B_s = t$, $B_p = P^t(x)$, and $B = (t, P^t(x))$.

1.2 Problem Description

The complete problem of constructing a Bayesian belief network from data has two distinct phases: topology construction and parameter determination. The topology construction phase has two aspects: assigning connections between different nodes and assigning direction. The METD2 algorithm will introduce hidden variables in the first part of topology construction, when assigning connections; indeed, the only connections possible are between hidden nodes and observable variables and a hidden node to other hidden nodes. The METD2 algorithm assumes that the hidden variables cause their children, whether a child is another hidden node or a leaf (observable) node. It uses this assumption to determine the probabilities of and correlation on the hidden nodes, which completes the second major phase of model construction.

Thus the problem is to construct from the database a hidden network structure with the maximum likelihood of being correct for that database, or alternatively, to construct a hidden network structure which has minimum entropy given the database. As noted by [Pearl, 1988; 382], a model with too many links is computationally useless because of storage requirements and propagation times. What structure-finding algorithms need is an inductive bias for simple, sparse structures. The METD2 structuring algorithm assumes the simple structure of a tree and then finds parameters to fit it.

A search of the literature shows several approaches to finding relationships among the observable variables, but only a few for finding relationships between variables in the database and hypothetical ones ("hidden" variables) not in the database. Since most approaches to finding structure (in the general case) are either computationally intractable, do not have normative halting criteria, or have computationally unfeasible running times, any new algorithm should have at worst polynomial time complexity and a clear halting mechanism. More importantly, the dearth of algorithms for finding hidden structure in data invites research into creating such algorithms.

Hence, the primary goal of this thesis is to define, develop, and evaluate a specific procedure to learn hidden structure from data. The METD2 algorithm will decompose a joint distribution of n observable variables into $n+1$ observable and hidden variables. The hidden variable will exist in the form of a tree consisting of $n-1$ nodes. The Tree Decomposition will construct a binary tree network with minimum cross-correlation error between any quadruple of observable nodes. The procedure will input a database of cases to derive all prior probabilistic knowledge about the domain. The final product of the METD2 procedure will be a combined tree whose n leaves are the observable variables in a sample and whose $n-1$ interior nodes are the marginalizations (hidden nodes) for the leaves. Since no work has been done to implement and test the Minimum-Error Tree Decomposition algorithm since [Liu et al, 1990] proposed it, this thesis shall encompass probability theory, algorithmic design, code implementation, and methodology evaluation. This thesis refers to the algorithm proposed by [Liu et al, 1990] as METD1.

1.3 Content Overview

In **Chapter 2**, I present the pseudocode for the METD2 algorithm, introduce some additional notation, and provide an example of how METD2 would generate a simple tree-structured hidden-valued belief network.

In **Chapter 3**, I review previous research that led to the development of METD2 as well as related research in the problem of finding structure. First, I examine Spanning Trees with Direction Discovery and Kutato/K2, two approaches which find associations between observable variables. Next, I examine a non-Bayesian method for learning—Artificial Neural Networks/Error-based gradient descent. Then, I examine two approaches to finding hidden associations, AutoClass and Exact Tree Decomposition. Finally, I consider an expansion of Exact Tree Decomposition to Minimum-Error Tree Decomposition (METD1), which led directly to the METD2 approach.

In **Chapter 4**, I first detail the acquisition and use of correlation and conditional probability in METD2. Then, I define the METD2 algorithm theoretically, including those points originally presented in [Liu et al, 1990]. Next, I describe algorithm implementation details important to making the algorithm computationally efficient. Last, I briefly consider the computational complexity of the final algorithm implementation.

In **Chapter 5**, I define the data-generation method used to create sample databases for testing the METD2 algorithm. Then, I present the results obtained from applying the algorithm to three artificial problem domains and one real-world problem domain. In analyzing the results, I will consider accuracy, efficiency, and qualitative performance, and provide a brief comparison

to K2, a related structuring algorithm. Finally, I present results on space, time, and storage of METD2.

In **Chapter 6**, I conclude this thesis with a discussion of the results, my contributions to belief networks, and future research areas in the problem of finding hidden structure. Some of the future research areas include neural network optimization, the meaning of the hidden variables discovered by METD2, and further generalization and expansion of the algorithm.

2. INTRODUCTION TO METD2 ALGORITHM

Chapter 1 provides an introduction to Bayesian belief networks and to the problem of finding hidden structure in Bayesian belief networks. In this chapter, I present an overview of the METD2 algorithm and show how it can be used to construct a hidden-valued tree-structured network. This chapter does not consider the details of the statistical or tree-construction processes. Chapter 4 considers the formal underpinnings and the implementation details. In this chapter, I first provide the pseudocode for the algorithm. Then, I trace through the METD2 algorithm in operation and show its outputs at each stage. Chapter 5 examines the experimental results from applying this algorithm, while Chapter 6 concludes this thesis.

First, the METD2 algorithm (Figure 2.1) reads into memory the database, $\boldsymbol{D}$, and performs statistical analysis to determine the correlation on all pairs of variables, $\boldsymbol{i}$ and $\boldsymbol{j}$. The variables in $\boldsymbol{D}$ are the observable variables in the problem domain. Using the correlation table, the algorithm determines the cross correlation error for all $3\binom{N}{4}$ quadruples and sorts the quadruples into order of increasing error. It then uses a greedy search to build the hidden-structure such that the cross-correlation error between any set of four leaves is minimal. Next, it uses matrix methods to determine the correlation between interior nodes and interior nodes and leaves. Last, it finds prior probabilities for the interior nodes and conditional probabilities for the

leaves on the interior nodes and the interior nodes on each other. The algorithm consists of 14 steps (shown in Figure 2.1). This thesis does not implement the last two steps (Steps 13-14).

procedure METD2:
{ *Input*: a database *D* with *n* variables per case }
{ *Output*: a tree *T* whose leaves are the *n* observable variables in *D* and whose interior }
{ nodes were added by the METD2 algorithm. }
begin
(1) Initialize the set of independent nodes, I, |I| = N, and the forest, F, F = ∅.
(2) If |I| < 4, return the star-decomposition for I.
(3) Define the correlation on I.
(4) Generate quadruples of four leaves, sort them by minimum cross-correlation error, and construct lists of quadruple heaps, K and Q'.
(5) **While** ((|I| > 0)) or (|F| > 1)) **do**
(6) Set min_1 to the minimum error quadruple in Q'.
(7) If (|F| > 1), set min_2 to the minimum error between any two trees in F.
(8) If (|F| > 0) and (|I| > 1), Set min_3 to the minimum error between any two leaves in I and any tree in F.
(9) If (|F| > 0) and (|I| > 0), Set min_4 to the minimum error between any leaf in I and any tree in F.
(10) Add the min(min1,min2,min3,min4) to a tree, T, in F, or to F, and remove used leaves from I, quadruples from Q', and/or trees from F.
end;
(11) Remove non-terminating nodes and consolidate the tree.
(12) Check the tree for consistency.
(13) Find correlation of interior nodes with each other and with leaves.
(14) Find prior and conditional probabilities for interior nodes.
end.

Figure 2.1 Pseudocode Listing for METD2.

In the remainder of this chapter, I run METD2 on an example database, ***D***, and show the output of each step of the algorithm. Figure 2.2 shows that ***D*** contains five samples of a domain with seven observable features.

	Feature 0	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6
Sample 1:	1	1	0	1	1	0	1
Sample 2:	0	1	0	1	1	0	1
Sample 3:	0	1	0	0	1	1	1
Sample 4:	0	1	0	0	0	1	1
Sample 5:	1	0	0	0	1	1	1

Figure 2.2 Domain Database for Example.

In Step 1, METD2 confirms that the network has more than 3 variables. In Step 2, METD2 initializes the set of independent leaf nodes, ***I***. Since ***D*** contains samples of a causal network with 7 observable variables, each integer in ***I*** will uniquely label a variable in the database, such that initially ***I*** = *{0, 1, 2, 3, 4, 5, 6}*. In Step 3, METD2 determines from the sample statistics the correlation on all pairs of variables in ***I*** (Figure 2.3).

Features	0	1	2	3	4	5	6
0	1						
1	-0.61237	1					
2	0	0	1				
3	0.16667	0.40825	0	1			
4	0.40825	-0.25	0	0.40825	1		
5	-0.16667	-0.40825	0	-1	-0.40825	1	
6	0	0	0	0	0	0	1

Figure 2.3 Correlation on Features in Domain Database.

In Step 4, METD2 builds a heap, Q, which contains the absolute value of the cross-correlation error for all sets of four nodes in I, sorted in order of ascending cross-correlation error (Figure 2.4). The algorithm also builds a "kill" list of heaps, K, and then copies Q into Q', also a list of heaps. Both K and Q' help reduce the computational complexity of the METD2 algorithm. We will ignore K and Q' for now, since their behavior does not affect the presentation of this example.

Heap Location	Correlation-Error	Quadruple Members
0, 0	0.00	<0 1 2 3>
0, 1	0.00	<0 1 2 5>
0, 2	0.00	<0 1 3 6>
0, 3	0.00	<0 1 2 4>
0, 4	0.00	<0 1 4 6>
0, 5	0.00	<0 1 5 6>
0, 6	0.00	<0 2 3 4>
⋮	⋮	⋮
0, 19	0.208	<0 1 3 4>
0, 20	0.208	<0 1 4 5>
⋮	⋮	⋮
1, 0	0.00	<1 2 3 4>
⋮	⋮	⋮
3, 1	0.000	<3 4 5 6>

Figure 2.4 Selected Items in the Quadruple Heap.

Since $|I| > 0$ we enter the loop of Steps 6-10. In Step 6, METD2 examines the heap of quadruples and removes the set of four leaves with minimum cross-correlation error. From Figure 2.4, we see that multiple entries have the same minimum error of 0.00, and the algorithm

selects the first item in combinatorial order. This is $\langle i\ j\ k\ l \rangle = \langle 0\ 1\ 2\ 3 \rangle$. In the first pass, $|F| = 0$, therefore METD2 will not execute Steps 7-9. In Step 10, METD2 reduces the set I to {4, 5, 6}, and the set $Q = \{\}$ since every element in Q contained a member of {0, 1, 2, 3}. It then makes the initial tree, shown in Figure 2.5:

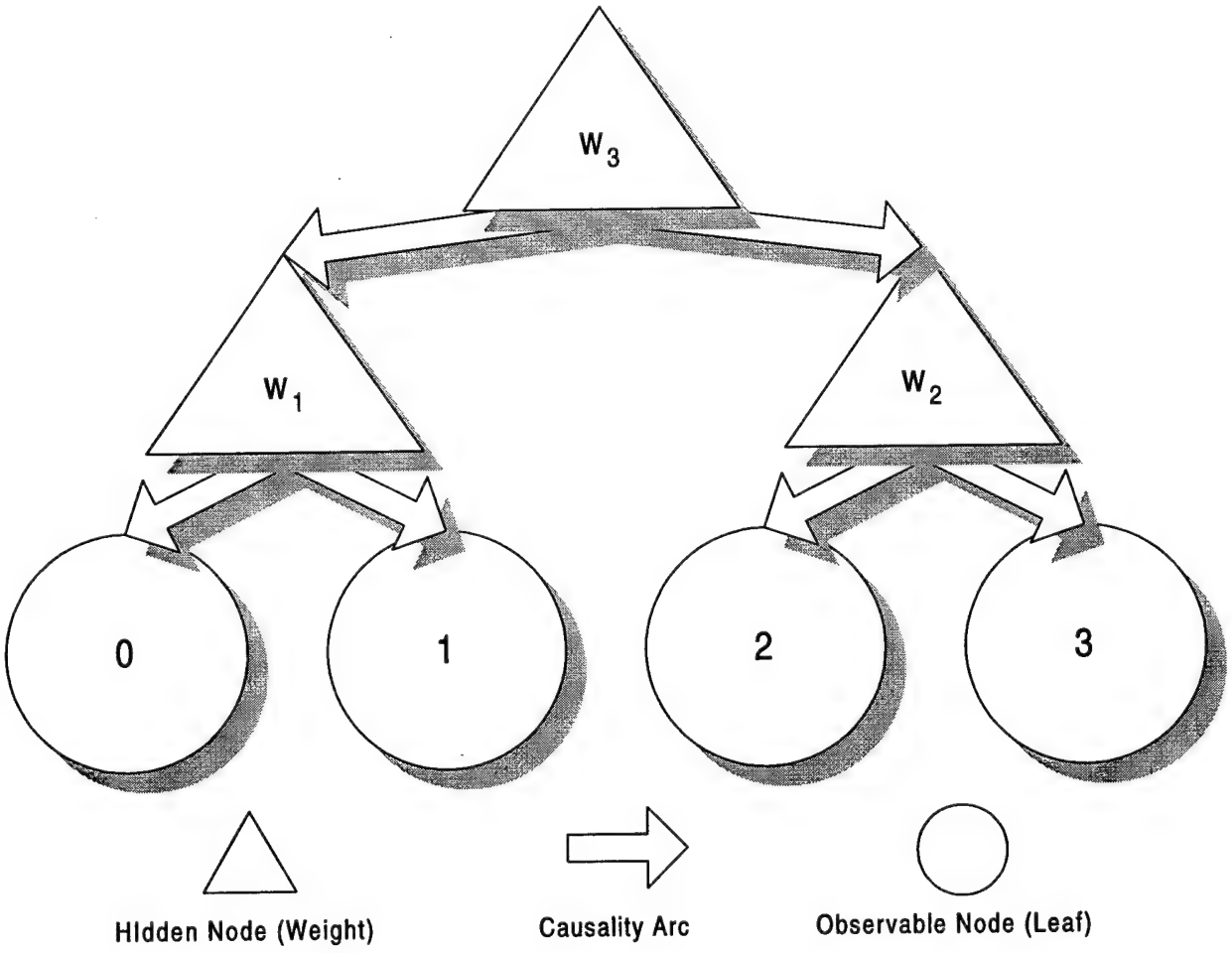


Figure 2.5 Initial Tree, Showing Combination of Two Independent Node Pairs (0,1), (2,3).

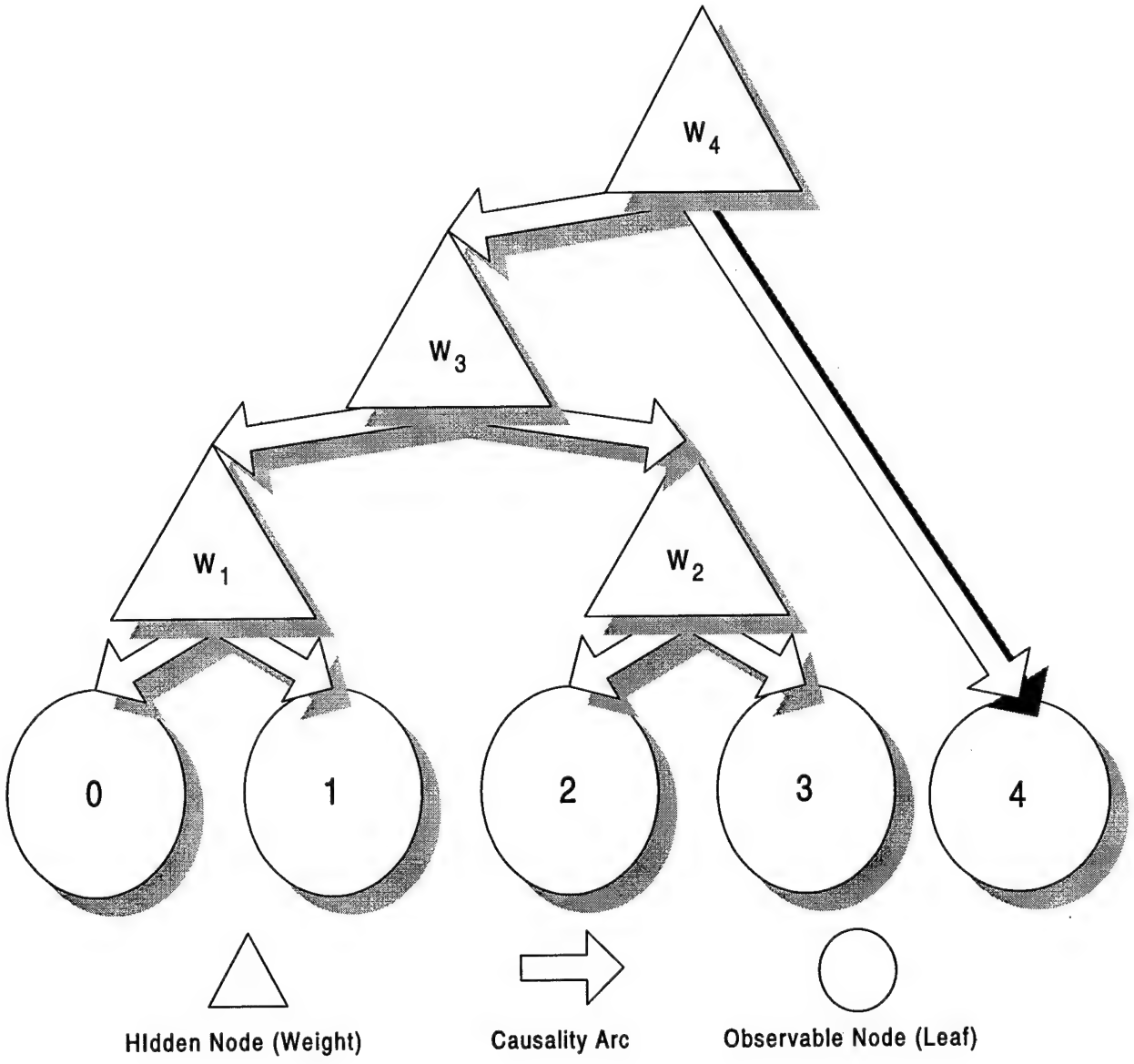


Figure 2.6 Modified Tree, Showing Combination with a Single Leaf (4).

In the second pass, $|I| = 3$ and $|F| = 1$. Since $|I| \leq 4$, METD2 does not execute Step 6. Since $|F| \leq 2$, the algorithm does not execute Step 7. From Step 8, METD2 finds that the minimum error between any two leaves in I and any two leaves from a tree in F. This is <(i j) (k l)> = <(0 1) (4 5)> with an error of 0.208. The algorithm also finds the minimum error between a

single node in I and any two leaves from a tree in F. This is <(i j k) l> = <(0 1 3) 4> with an error of 0.208. Looking at more significant figures, Step 9 does yield a lower error than Step 8, and the global minimal error is the same as that found in Step 9. Step 10 constructs a new tree (Figure 2.6) and removes {4} from I, and Q = { } from the first pass.

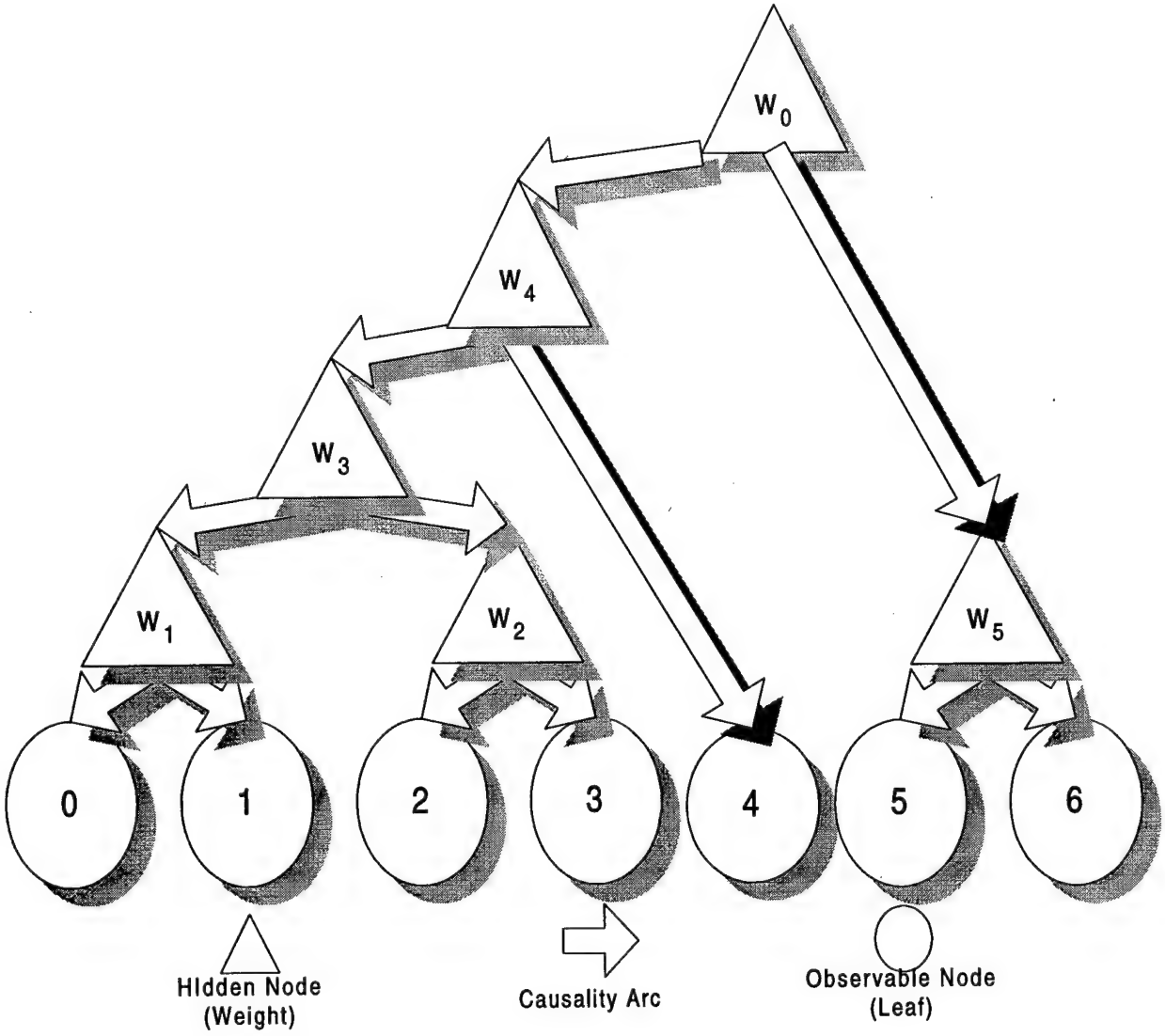


Figure 2.7 Final Tree, Showing Combination with a Leaf Pair (5,6).

In the third pass, since $|I| = 2$ and $|F| = 1$, the algorithm will not execute Steps 6-7. Step 8 yields the minimum error combination of <(0 1) (5 6)> with an error of 0.00, while Step 9 yields the minimum error combination of <(0 1 4) 5> with an error of 0.208. Step 10 finds the global minimum error as that found in Step 9, 0.00. In Step 10, $I = I - \{5,6\} = \{\}$, and $\mathbf{Q} = \{\}$ from the first pass. The resulting tree is the illustration shown in Figure 2.7.

In the fourth pass, the loop terminates at its start because $|I| = 0$ and $|F| = 1$. The algorithm then executes Steps 11-12, which in this example, yields the tree shown in Figure 2.7, since no additional changes were necessary. Steps 13-14 will then perform the next phase of determining the prior and conditional probabilities for the interior nodes that Steps 1-12 just constructed.

3. PREVIOUS AND RELATED RESEARCH

Chapters 1 and 2 provide, respectively, an introduction to Bayesian belief networks and the METD2 algorithm for finding hidden structure. In this chapter, I examine some of the research into the problem of finding a Bayesian probabilistic model given a sample database. Since the majority of approaches to finding structure discover associations between observable variables, I first examine two such methods—Spanning Tree Construction with Direction Discovery and Kutato/K2. The ideas introduced while examining Spanning Tree Construction and K2 will help motivate the discussion of hidden structure discovery in the second half of this chapter, and lead, in Chapter 6, to some ideas for combining the two classes of algorithms. The second half of this chapter surveys the AutoClass project and then examines, in some detail, Exact Tree Decomposition. Both AutoClass and Exact Tree Decomposition specifically attempt to find a hidden structure for a set of observed variables. The last Section of this chapter, Section 3.5, examines the work of [Liu et al, 1990] on a generalization of Exact Tree Decomposition, Minimum-Error Tree Decomposition I (METD1). METD1 is the direct predecessor of METD2. Chapters 4-6, respectively, provide the detailed algorithmic specification, experimental results from my METD2 implementation, and then, a discussion of the results and the conclusion to this thesis.

3.1 Related Bayesian Methods: Learning Structure from Complete Data

Each of the algorithms described in this section (and Section 3.3, too) constructs a directed-acyclic graph (DAG) structure, since evaluating a Bayesian belief network and obtaining an exact solution, in the general case with cycles, is NP-hard [Charniak, 1991] and [Pearl, 1988]. For the special case of causal polytrees, or singly-connected networks, the network can be evaluated in time linear to the number of nodes [Pearl, 1988], [Neopolitan, 1990]. For the even more restricted class of trees, the network can be evaluated in time linear to the natural log of the number of nodes. [Pearl, 1988] is the chief pioneer of research into d-separation and the necessary and sufficient conditions for polynomial and linear time network evaluation and polynomial time network construction.

3.1.1 Spanning Trees with Direction Discovery

3.1.1.1 Maximum Weight Spanning Tree (MWST)

To understand the flavor of the algorithm which we will eventually construct, it will be helpful to understand an early algorithm to construct a BBN from data developed by [Chow & Liu, 1966]. This algorithm differs from the one we will construct in that it does not consider the presence of hidden structure but assumes that all the observable variables are present in the database.

For selection, or distance, measure, [Chow & Liu, 1966] use the Kullback-Leibler cross-entropy:

$$I(X_i, X_j) = \sum_{X_i, X_j} P(X_i, X_j) \log \frac{P(X_i, X_j)}{P(X_i) P(X_j)} \geq 0.$$

Note that this measure generates values between 0 and 1, and generates 0 if the two nodes, $\mathbf{X_i}$, $\mathbf{X_j}$, are completely *independent* and 1 if the two nodes are completely *dependent.*

Figure 3.1 shows the steps of the Maximum Weight Spanning Tree. In Step 1 of Figure 3.1, MWST calculates the joint probabilities for the variables contained in the source database *D*. In Step 2, it uses the Kullback-Leibler cross-entropy measure to compute the weights of each branch (a joining of two nodes). In Step 3, it picks the "largest" branch (the one with the highest cross-entropy measure) and begins to construct the tree. The loop of Steps 4-5 repeat the process started in Step 3, until there are no branches remaining. The final result is a tree whose connected nodes (branches) have maximal dependency ("weight").

procedure MWST:
{ *Input*: a database *D* with *n* variables per case }
{ *Output*: a polytree *P* whose nodes are the *n* observable variables in *D and whose* }
{ *connections* were added by the MWST algorithm. }
begin
(1) Compute the joint probability distribution $P(x_i,x_j)$ for all pairs, given observed $P(x)$.
(2) Using the joint distributions from (1), compute all n(n-1)/2 branch weights.
(3) Initialize the tree from largest two branches (in magnitude) computed in (2).
(4) **While** (branches remain) **do** { loop will execute *n-1* times }
(5) Take the next largest branch (from (2)) and add to the tree if it does not form a loop.
(6) $P^t_p(x)$ is simply the chain rule (Equation 3) applied to an arbitrary root node.
end.

Figure 3.1 Pseudocode listing for MWST.

At each repetition of Step 5 of Figure 3.1, MWST only adds the branch that has the greatest dependency between its two members. Like other heuristic searching methods (such as METD2), the MWST approach uses a greedy-search which locally maximizes an objective function. For MWST, the objective function is the Kullback-Leibler cross-entropy.

3.1.1.2 Incomplete Direction Discovery

While the MWST algorithm will construct a tree-structured graph given a database, it does not find the *directions* of the connections in that graph. [Pearl, 1988] provides a direction discovery method to determine most of the directions of the arcs in a directed-acyclic graph. The algorithm in Figure 3.3 works by examining the causal basins present in the DAG and determining direction (Figure 3.2) for the leaves of the basin and then working inward from the leaves to the interior nodes.

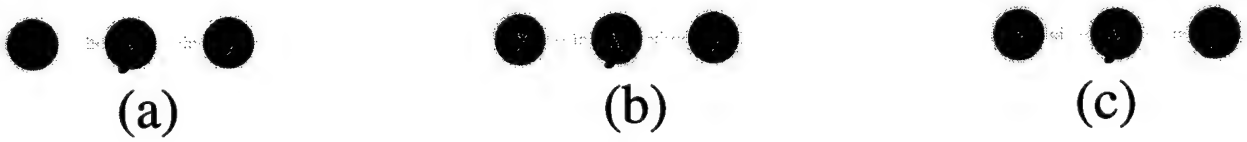


Figure 3.2 XYZ Test for Polytree Structuring.

The XYZ test of Figure 3.2, called in the Direction-Discovery algorithm of Figure 3.3 is to test, using the joint and conditional probability tables, whether X is conditionally independent of Y given Z and whether Y is conditionally independent of X given Z. Figure 3.2 shows the topologies distinguished by this test. Since the configurations illustrated in (a) and (b) are not distinguishable using the XYZ test, some of the arcs in the resulting graph will be undirected, and the user of the algorithm has to supply external semantics to define the relationship.

procedure Direction-Discovery:

{ *Input*: an acyclic graph *G* and a database *D*, each with *n* variables }

{ *Output*: a polytree *P* whose nodes are the *n* observable variables in *D and whose* }

{ were determined by the Direction-Discovery algorithm. }

begin

(1) Search in the internal nodes of the skeleton, beginning with the leaves and working inward, until a multi-parent node is found using the XYZ test.

(2) **While** (directionality remains to be discovered) **do**

(3) When a multi-parent node *n* is found, determine the directionality of all of its branches using the XYZ test.

(4) For each node having at lest one incoming arc, resolve the directionality of all of its remaining adjacent branches using the XYZ test.

End;

(5) If there remain undirected branches, label them "undetermined" and supply external semantics.

end.

Figure 3.3 Pearl's Direction-Discovery Algorithm.

While the MWST algorithm specified in Figure 3.1 creates a belief network without direction, Pearl's direction-discovery algorithm of Figure 3.3 can partially assign direction to an acyclic graph. Thus, to combine the two algorithms, we simply run Pearl's Direction-Discovery method on the output from the MWST procedure. As a last step, the user supplies external semantics for any arcs Pearl's algorithm cannot label.

3.1.2 Kutato/K2

[Cooper and Herskovits, 1992] describe a more complicated algorithm family, Kutato/K2, for finding the associations between observable variables. Their metric for maximum likelihood performs a heuristic search over the space of all possible belief networks which could possibly fit some data structure to find the most likely of the possible structures. They have shown that K2 accurately finds the relationships between observable variables in a sample biomedical database called *Alarm*. The *Alarm* database contains the diagnostic data from running medical tests on a life-like diagnostic simulation of a medical patient. Chapter 5.2 shows two belief networks constructed from this database.

[Herskovits, 1991] identifies three major approaches to constructing a belief network, B, which consists of structure and probabilities, (B_s, B_p), from a database, D. Each approach attempts to maximize the likelihood of the constructed network model, $P(B_s, B_p)$, given the data. This likelihood maximization is equivalent to minimizing the entropy, I, for the model. The approaches are

1. To maximize $P(B_s, B_p \mid D)$ over B_s, B_p simultaneously.
2. To maximize $P(B_s, B_p \mid D)$ over B_s, then maximize $P(B_s, B_p \mid D)$ over B_p.
3. To maximize $P(B_s, B_p \mid D)$ over B_s, then acquire the conditional probabilities, B_p, directly from the database, D.

[Herskovits, 1991] notes that the first method is the most computationally expensive, and his two algorithms, Kutato and K2, both follow the third approach. Like Kutato and K2, METD2 uses the third approach since it is computationally less expensive and simpler than the prior two. As

noted earlier in the overview of Chapter 1, and later clarified in the overview to Chapter 4, using the third approach requires that several assumptions be made about the data. Recall that the Bayesian methods described in this section, as well as METD2, assume that the samples contained in a database are independent of each other—that is, the data-collection process did not bias the data. These Bayesian learning methods also assume that the variables in the sample can be fit to a uniform distribution. METD2 attempts to fit this data to a tree-dependent hidden-structured distribution, while other methods do not make assumptions, or else, require that assumptions do not be made, about the underlying of the problem domain.

3.2 Non-Bayesian Methods for Learning—Artificial Neural Networks

The related field of artificial neural networks has several statistical regression methods which are both fast and optimal for real-world data. The most popular of these methods is error-based back-propagation, as described in [Rumelhart et al, 1986]. Since there is a large volume of research into artificial neural networks, the METD1/2 approach may benefit from applying these techniques. For example, for continuous variables, the logistic or hyperbolic tangent functions provide easy-to-determine derivatives and thus provide fast and efficient parameter optimization.

A network with a single hidden layer, or *general classifier*, is able to model any function, provided the ANN possesses enough elements [Hassoun, 1995]. Figure 3.4 illustrates a processing element. Since the BBN usually assumes a Boolean AND or OR type of joint probability it may not be able to model effectively every type of function. Joining it to an ANN system should upgrade its predictive/classifying performance. Given enough processing elements and enough training examples, the ANN will optimize the toughest domain problems.

The chief disadvantage with this capability is that there is no way to determine *a priori* the number of elements required, or the number of training steps necessary, to converge to a solution with a desired degree of accuracy. A second difficulty is with the ANN-BBN combination: the ANN requires supervised learning while most BBN methodologies are unsupervised. The METD2 algorithm (an unsupervised method) would require extensive modification to handle error-based back-propagation.

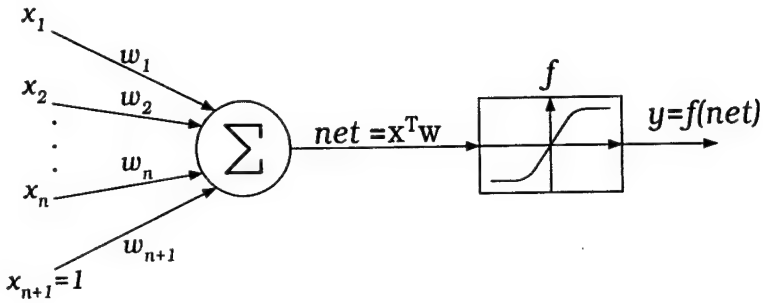


Figure 3.4 Processing Element.

Also an ANN, once trained, executes faster than other systems developed through learning methods. For real-time, real-world use, fast execution is essential. While the BBN architecture developed through METD2 is compact and fast, learning for BBNs is generally quite slow. An ANN learning method may permit real-time learning. If the computer-constructed BBN is implemented in a mission critical area, such as military battlefield awareness or corporate market forecasting, slow updating/learning of model parameters could lead to mission failure or corporate financial loss.

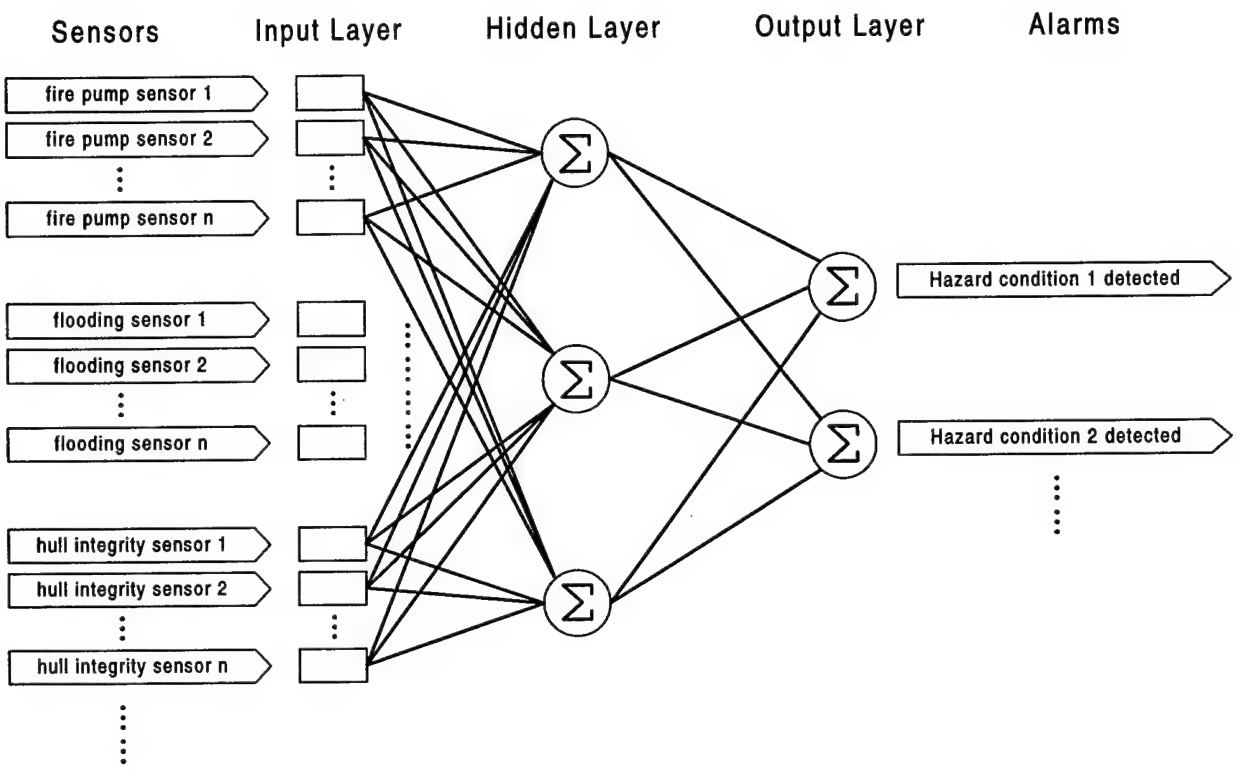


Figure 3.5 An Artificial Neural Network.

The basic ANN topology consists of three layers—an input layer which receives some pattern, a hidden layer which does intermediate processing on the input pattern vector, and an output layer which does final processing and outputs the classification of the input pattern. Each element in the hidden layer acts as summer, applying a unique function on the input vector, while the output layer treats the hidden layer's outputs as input and in turn applies a unique function on the hidden layer outputs. Figure 3.5 provides an illustration in the domain of ship damage control. In Figure 3.5, we see that the inputs to the network consist of multiple sensors—compartment smoke-detectors, fire-pump sensors, flooding sensors, hull integrity evaluators, environmental monitors, and etceteras—while the output layer indicates the presence or absence

of various alarm conditions, such as fire incipient, flooding incipient, or loss-of-balance incipient.

3.3 Previous Bayesian Methods for Learning Hidden Structure

Researchers have developed several metrics for performing belief network learning and calculating the data marginal likelihood. For example the AutoClass scoring function [Cheeseman and Stutz, 1989], the K2 or Bayesian Dirichlet criterion [Cooper and Herskovits, 1992], and the Bayesian Information Metric/Minimum Description Length (BIC/MDL) scoring function [Draper, 1993]. METD2 adds a new metric, minimum cross-correlation error, as described in Chapter 4. Each of these methods has been modified to find hidden structure from data, as examined in [Chickering and Heckerman, 1996]. Note that Chickering and Heckerman define *complete data* as data for which all observations are available, and *hidden variables* as a subset of missing observations in general. That is, a variable which never gets observed is *hidden*. Using Chickering and Heckerman's terminology, each of the learning methods listed above fall within the class of methods termed *asymptotic approximations*, which build and test Bayesian belief domain models. [Heckerman, 1995; 39] notes that there are very few experimental results available for learning Bayesian networks, but among these he lists [Aliferis and Cooper, 1994], [Spirtes and Meek, 1995], [Heckerman et al., 1995], [Chickering, 1996], and [Madigan and Raftery, 1994].

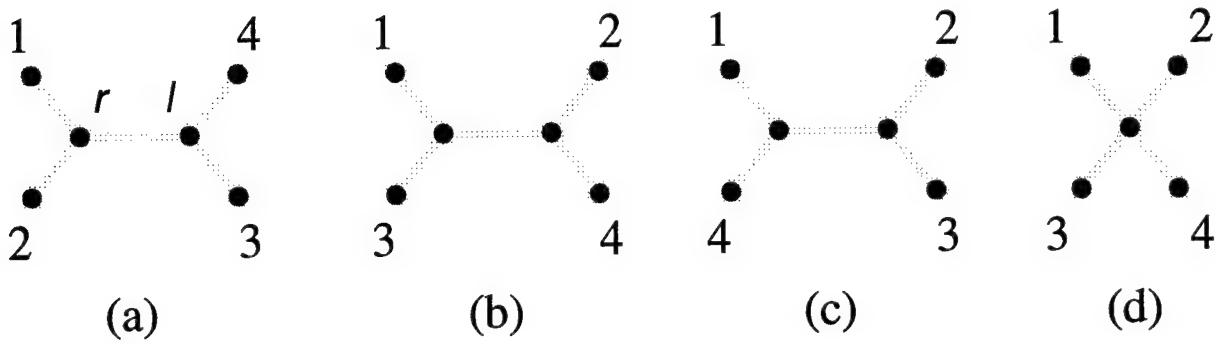
3.3.1 AutoClass

While MWST with Direction Discovery and Kutato/K2 do not directly find a hidden structure, AutoClass determines a kind of hidden structure by conditioning a set of observations on a single hidden variable. In a sense, this approach is more inline with the Artificial Neural Network school than Bayesian models because the user does not directly provide any semantics. Thus AutoClass and METD2 both fall within the class of techniques which use a "predetermined" graphical structures and simply condition all data in the set on one, or a small set, of hidden variables.

AutoClass has already been successfully applied to classifying NASA's astral spectrometry database, and because it found several new classes never noticed by human experts, its results form the basis of a new spectrometry catalogue. The AutoClass system performs a kind of unsupervised learning on data, where the algorithm attempts to classify a given database into the likeliest number of categories [Cheeseman et al, 1989].

3.3.2 Exact Tree Decomposition

To begin the process of developing a tree-structure for hidden variables, we need to determine which of four possible relationships may exist between any set of four variables within the provided data. [Pearl, 1988] identifies four unique structural relationships for connecting four leaves (observable variables) into quadruples, as shown in Figure 3.6:



$$p_{13}p_{42} = p_{14}p_{32} \qquad p_{12}p_{43} = p_{14}p_{23} \qquad p_{12}p_{34} = p_{13}p_{24}$$

Figure 3.6 Four Possible Topologies for Quadruples.

These four possible topologies give rise to equations of the form

$$p_{13} = p_{1r}p_{rs}p_{s3}, \qquad p_{42} = p_{4r}p_{rs}p_{s2}, \qquad \text{etc.}$$

For Tree Decomposition, all equations should hold at the same time (the fourth structure in Figure 3.6), which yields the sufficient precondition for topology construction of

$$p_{ik}\,p_{jl} = p_{il}\,p_{jk} \tag{5a}$$

as shown by [Pearl, 1988]. The above equation is equivalent to

$$d_{(i,j)\setminus(k,l)} = p_{ik}\,p_{jl} - p_{il}\,p_{jk} = 0. \tag{5b}$$

where d is the distance between the two pairs (i,j) and $(k.l)$. We will use (5b) in Chapter 4 as the basis of the minimization approach to Tree Decomposition.

The main problem with [Pearl, 1988]'s approach to Tree Decomposition is that it is too restrictive. (1) The approach requires exact knowledge of the correlation coefficients—there must be an perfect match between p_{ij} and p_{kl} for the algorithm to work. (2) It works only for data for which the underlying model is in fact tree-decomposable. (3) Simply relaxing the $p_{ik}\,p_{jl} = p_{il}\,p_{jk}$ equality for minimal difference is sensitive to inaccuracies in the data because there is not

method to restrict earlier branch decisions if correlation later shown inaccurate. Therefore, (4) placement of the $(i+1)^{th}$ leaf should be decided by its relation to all previously structured triplets sharing w as a center—which will substantially increase the complexity of the algorithm. The need to handle data with errors in the correlation coefficient (limitations (1) and (3)), or which may not be innately tree-decomposable (limitation (2)), leads to the development of the METD2 algorithm, which will handle the complexities noted by (4).

[Cooper, 1995] has also proposed a constraint-based approach to learning hidden structure from data. The particular set of assumptions in Exact Tree Decomposition is one of several possible sets of assumptions that he identifies. Among Cooper's other constraint sets may be a solution which is more general than Exact Tree Decomposition, but this is not clear from [Cooper, 1995].

3.3.3 METD1

The first generalization of Exact Tree Decomposition appears in [Liu et al, 1990], where Liu modifies the zero-difference cross-correlation error formula of [Pearl, 1988] (Equations 5(a) and 5(b)) to a minimum-difference cross-correlation error function (Equations 6-9 of Chapter 4). Pearl's contribution was that he determined a general approach to finding hidden structure when the correlation coefficients of the observed variables are exact and the underlying domain model is tree-decomposable. Liu's contribution was that he generalized Pearl's solution to noisy data, where the correlation coefficients are inexact, and to domains which may not be inherently tree-decomposable. Thus, Liu's generalization of Pearl's formula searches through the space of possible tree-structured Bayesian networks for a given database and finds the closest tree-

structured approximation. Neither Pearl nor Liu implemented their proposed algorithms. Thus, by designating Liu's work as METD1, I wish to distinguish his proposal for an algorithm from my implementation of that algorithm, METD2. Hence, the first place where this thesis differs from the previous research into Minimum Error Tree Decomposition is that it actually tests and implements an algorithm. This implementation is the basis of the experimental results of Chapter 5 on real and synthetic data sets.

4. METD2 ALGORITHM

Chapters 1 and 3, respectively, provide and introduction and a description of previous research, while Chapters 2 provides a high-level introduction to the METD2 algorithm. In this chapter, I examine the lower-level details that lie behind implementing each step of the METD2 algorithm. Thus, I make many references to the Figure 2.1 of Chapter 2, which provided the overall algorithm. The first section of this chapter, Section 4.1, describes the process of gathering the prior probabilities and the conditional probabilities from data. The next section of this chapter, Section 4.2, deals with the definition of the minimum cross-correlation error metric, the definition of the graphical topology combination operations, and the definition of the data structure reduction operations. Section 4.2 also specifies the component algorithms which implement each step of the METD2 algorithm, thus providing the transition between the mathematical definitions of Section 4.1 and the high-level algorithmic description of Chapter 2. This chapter also details the computational-complexity analysis, in Section 4.3, which shows that the worst-case time-complexity for the METD2 algorithm is $O(N^5)$ in the number of observable variables, n, found in each data sample, *not* the number of cases. The next chapter, Chapter 5, provides experimental results for the METD2 algorithm, and Chapter 6 will conclude this thesis.

The METD2 algorithm possesses three main phases, as shown in Figure 4.1—gathering the variables' correlation from the database, constructing the topology, and determining the network parameters. My implementation fits both discrete and continuous data to a normal

distribution, since the METD2 algorithm does not require the observable variable's conditional probabilities. Later implementations may expand to include the conditional probabilities because the parameter determination phase, as proposed by Liu, requires them. The topology construction phase makes use of a list of buckets of heaps of quadruples to optimize the algorithm's running time. Topology construction is the heart of the METD2 algorithm, and it is the focus of this research. The third phase of determining network parameters relies upon non-linear programming techniques of matrix manipulations. It was not implemented for this thesis. Last, I analyze the computational complexity as it relates to my implementation of METD2 and show that it has polynomial time computational complexity.

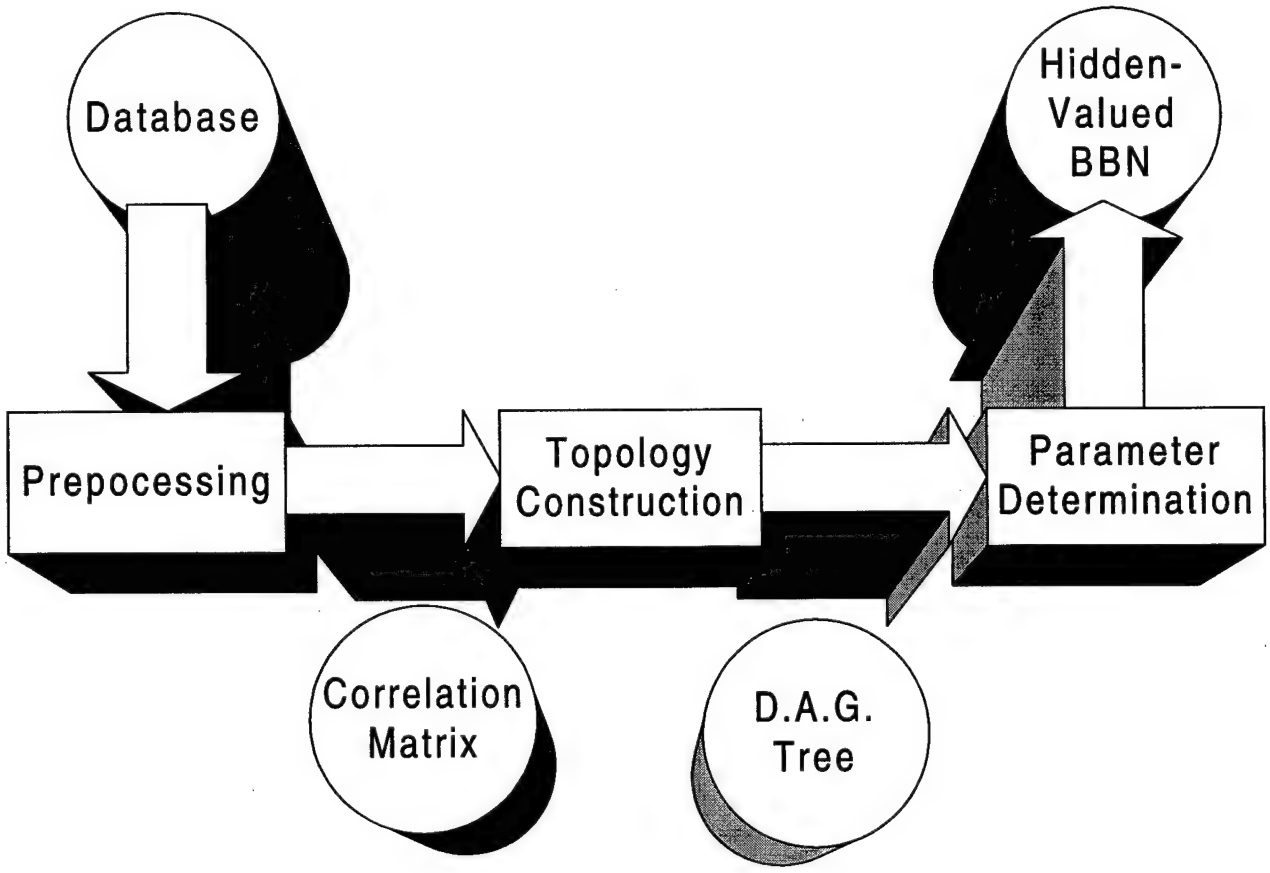


Figure 4.1 Three Phases of the Complete Algorithm.

4.1 Acquiring Correlation and Conditional Probability

Figure 4.1 shows that the input of the METD2 algorithm is an entire database of cases. Many learning algorithms, such as those used to train artificial neural networks, use an incremental update process were only one item from the entire database need be read at a time. As implemented, METD2 reads the entire database into memory to calculate sample statistics during the preprocessing phase. Therefore, it access the entire database at once. By batch-processing the data, the algorithm gathers the complete statistics available from the sample and generates a hidden structure with the minimum possible cross-correlation error among any four of its leaves. Since data usually contains errors the correlation will not be exact, but larger samples should contain smaller errors in the correlation coefficients of the variables.

The data for which we would like to generate models generally falls into two main classes: discrete and continuous. For both types of data, I make the following assumptions:

- The density function is uniform.
- Cases occur independently of each other given a BBN model.
- No case has missing observable variables.

Furthermore, I treat the discrete data as if they were continuous, and approximate both discrete and continuous data using a normal distribution. I chose this particular approach because I did not need to store the conditional probabilities for discrete data in a discrete format in order to determine the correlation coefficients for that data. In addition, it is not clear whether the naïve approach to extending the METD2 approach to non-Bernoulli, discrete data will work. Since prior work has been done on determining conditional probabilities for normal and Gaussian

continuous distributions, it seems appropriate for this thesis to treat all data as if it were samples drawn from an underlying continuous distribution.

The opposite approach is to discretize the continuous variables and treat both continuous and discrete data as if they were samples of a multinomial distribution. One disadvantage of this approach with respect to the continuous variables is that determining the number of classes and the ranges of each discrete class for the continuous random variable can be complex. One advantage is that the Dirichlet distribution (with parameters $\alpha_1, \ldots, \alpha_r$) provides an easy way to record the multinomial distribution contained in the data.

In general, the preprocessing stage will gather the $P(D|\theta)$, where D is the sample (database) and θ is the random variable about which we wish to gather data. The formula for the Dirichlet distribution (α), when used to estimate the probability of a particular random sample, is

$$P(\theta|D) = \frac{\Gamma(\alpha+n)}{\prod_{k=1}^{r}\Gamma(\alpha_k+n_k)}\prod_{k=1}^{r}\theta_k^{\alpha_k+n_k-1},$$

where n is the size of the random sample. When used to estimate a discrete random variable, the Dirichlet distribution reduces to the following equation

$$P(\theta) = c\prod_{k=1}^{r}\theta_k^{\alpha-P(x^k)-1},$$

where c is some normalization constant. Likewise, we follow a similar process to get the physical probability for a continuous random variable, that may be continuous or Gaussian. If (u, v) are the mean and standard deviation for a continuous function, the Gaussian formula is

$$P(x) = \frac{e^{-\frac{(x-u)^2}{2v}}}{\sqrt{2\pi v}}.$$

As noted, the procedure to gather statistics is simply to treat each variable as if it had a complete, uniform distribution, and then fit the gather statistics to a normal distribution.

4.2 Topology Construction

The METD2 algorithm uses a greedy search to minimize an error function, ***d***, for each local neighborhood of leaves (observable variables). Together, the leaves form a tree structure. Section 4.2.1 defines the tree structure, a graphical and mathematical construct which forms the foundation for the METD2 approach. Section 4.2.2 provides the mathematical definition of the cross-correlation metric for each possible topological combination permitted by METD2. Section 4.2.3 provides the graphical and mathematical specification for the combination and reduction operations, processes which permit an iterative approach to growing and then shrinking the final tree structure.

[Pearl, 1988] proposed the original algorithm for Exact Tree Decomposition and [Liu et al, 1990] describes a generalization that handles noisy input data. Neither Pearl nor Liu implemented their algorithms or provided an algorithmic specification. While Pearl limits his analysis to cases of exact trees, Liu's approach finds an approximate solution by using a heuristic-based greedy-search to minimize the cross-correlation error among different topological configurations. This thesis is the first implementation of Liu's and Pearl's Exact Tree Decomposition/Minimum-Error Tree Decomposition approaches to finding hidden structure.

4.2.1 Definition of a Tree Structure

A *tree structure*, as created by application of the METD2 algorithm to a database of cases, $\boldsymbol{D}$, is a binary tree, $\boldsymbol{T}$, which possesses the following properties: (1) the leaves of $\boldsymbol{T}$ are the n observable variables in $\boldsymbol{D}$; (2) the n-1 interior nodes in $\boldsymbol{T}$ marginalize the n leaf nodes; and (3) the n-1 interior nodes were added by the METD2 algorithm. An interior node *marginalizes* a pair of leaves, if the leaves are conditionally independent of each other given the interior node. The leaf nodes are labeled from 0 to n-1, and the interior nodes are labeled w_i with $\{1 \leq i \leq n\text{-}2\}$, with w_o referring to the root of the binary tree $\boldsymbol{T}$. Leaves, I_i, which are not members of tree $\boldsymbol{T}$ but are in $\boldsymbol{D}$, are called the *independent leaf nodes* or the independent nodes or the observable variables or the observable nodes. The interior nodes w_i are also called the hidden nodes or the *hidden variables* or the *weights* or the marginalization of the observable variables.

4.2.2 Definition of Decomposition Errors

The heart of the METD2 algorithm is minimizing the cross-correlation error between quadruples of nodes. When decomposing a distribution of n observable variables, we find that there are four possible topological cross-correlation errors to minimize.

- Definition 1. The cross-correlation error between two disjoint node pairs $(i, j)\backslash(k, l)$ is

$$d_{(i,j)\backslash(k,l)} = |p_{ik}\, p_{jl} - p_{il}\, p_{jk}| \tag{6}.$$

- Definition 2. The cross-correlation error between a tree w and a node pair (i, j) is

$$d_{(i,j)\backslash w} = d_{w\backslash(i,j)} = \max_{i,j\in l, i\neq j; k,l\in w, k\neq l} |p_{ik}\, p_{jl} - p_{il}\, p_{jk}| \tag{7}.$$

- Definition 3. The cross-correlation error between two trees w_1 and w_2 is

$$d_{w_1 \setminus w_2} = d_{w_2 \setminus w_1} = \max_{i,l \in w_1, i \neq j; k,l \in w_2, k \neq l} |p_{ik}\, p_{jl} - p_{il}\, p_{jk}| \quad (8).$$

- Definition 4. The cross-correlation error between a tree ***w*** and single leaf node ***i*** is

$$d_{i \setminus w} = d_{w \setminus i} = \max_{j,k,l \in w_1, j \neq k \neq l} |p_{ik}\, p_{jl} - p_{il}\, p_{jk}| \quad (9).$$

- Definition 5a. The minimum cross-correlation error between two node pairs ***(i, j)***, ***(k, l)***, or between a node pair ***(i, j)*** and a pair of leaves ***(k, l)*** from a tree ***wi***, is

$$\min\left(d_{(i,j)/(k,l)}\right) = \min\left(d_{(i,j)/(k,l)}, d_{(i,k)/(j,l)}, d_{(i,l)/(j,k)}\right) \quad (10a).$$

- Definition 5b. The alternative minimum cross-correlation error between two node pairs ***(i, j)***, ***(k, l)***, or between a node pair ***(i, j)*** and a pair of leaves ***(k, l)*** from a tree ***wi***, is

$$\frac{\sum_{i,j,k,l} d_{(i,j)/(k,l)}}{unique\left(\langle i,j,k,l \rangle\right)} = \frac{\sum_{i,j,k,l} d_{(i,j)/(k,l)}}{3}, \langle i,j,k,l \rangle \in T \quad (10b),$$

Definition 5 lists two alternatives to finding a particular maximum error between two node pairs ***(i, j)***, ***(k, l)***, or between a node pair ***(i, j)*** and a pair of leaves ***(k, l)*** from a tree w_i. While I use the minimum convolution error (Equation 10a) in Definition 5a, [Liu et al, 1990] uses the average convolution error (Equation 10b) in Definition 5b. I prefer the minimum instead of average cross-correlation error because it reduces the number of calculations necessary (by approximately 25%) and decreases the algorithm's running time (by approximately 5%-10%). In addition, it seems more correct, conceptually, than the average error approach because "minimum" error, in the case of the leaves, should refer to the true minimum, and not the average.

4.2.3 Definition of Combination and Reduction Operations

The guiding principle behind METD2 is to perform a locally-optimized greedy search. Each pass through the main loop of the algorithm (described in Figure 2.1) creates a new structural element such that the new element has minimum cross-correlation error with prior structural elements. If each element has locally-minimal error with all prior structural elements, then the entire tree has minimum cross-correlation error, and by that metric, is the optimal hidden structure for the data. This section thus defines the possible topological structures which METD2 can create or modify in a given iteration of the main algorithm. Each structure consists of a pair of two smaller structures—the left and right children of the whole. Thus, each combination of structures, or addition of a structure, has the property that the newly-created root marginalizes its children. Marginalization is the process of making the two nodes conditionally independent of each other given a third node. This section also describes the reduction operations necessary to maintain the data structures which hold the information necessary for the combination operations. Implementing the data structures efficiently is important because they control the computational complexity for the algorithm. Likewise, they must be accurate, or the METD2 algorithm will not work correctly.

The example METD2 application described in Chapter 2 illustrated three of the four possible tree combinations (tree to leaves, tree to leaf, and leaves to leaves). The fourth type of combination is tree to tree. Figures 4.2-4.5 show how each of these combinations constructs or mutates a tree. [Liu et al, 1990] provides the four definitions for the four types of combination operations which the METD2 algorithm might perform:

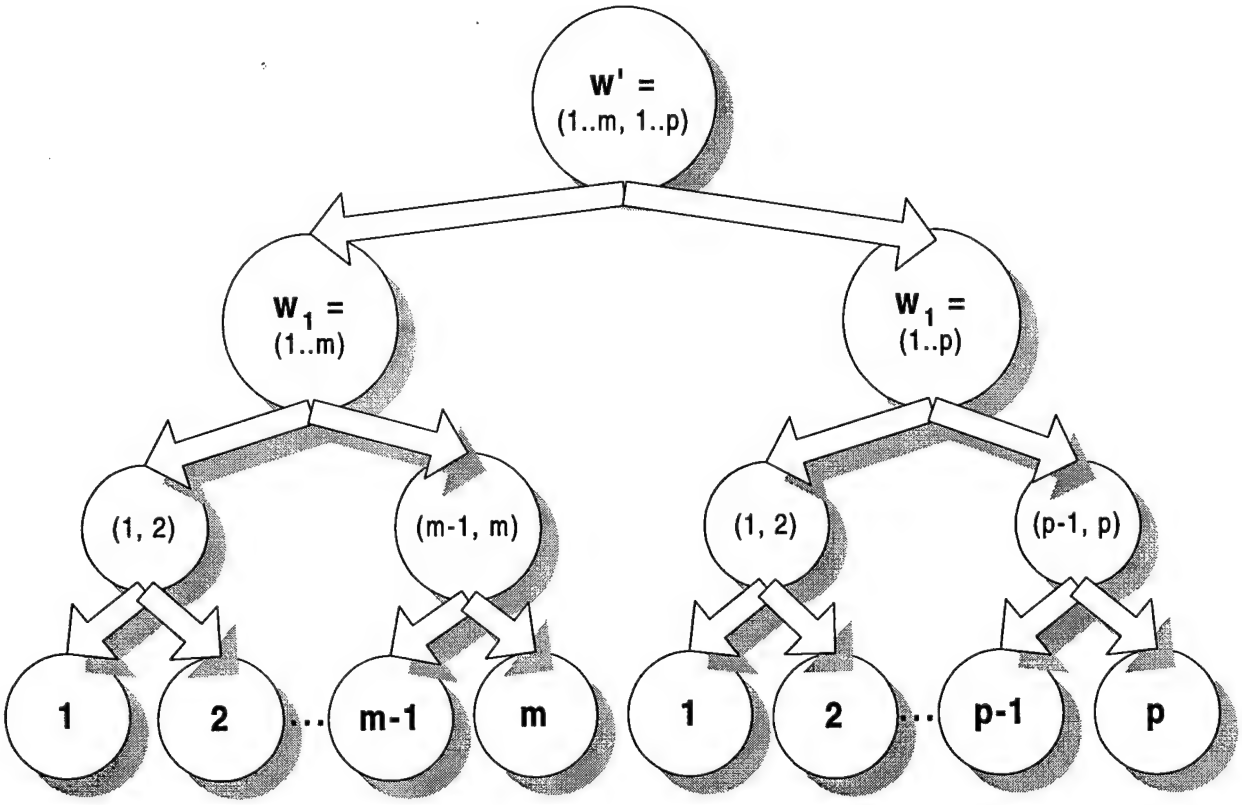


Figure 4.3 Combination of Two Trees—A Tree-Tree Join.

- Definition 7. The combination (join) of two trees w_1, w_1 = (1, …, m), and w_2, w_2 = (1, .., p), is a new tree, tree $T_{w1.w2} \mid w_1$ = value(T.left) and w_2 = value(T.right) and w' = value(root) = (1, .. ., m, 1, .., p), and I will refer to this type of combination operation as a tree-tree join. Figure 4.3 illustrates this definition.

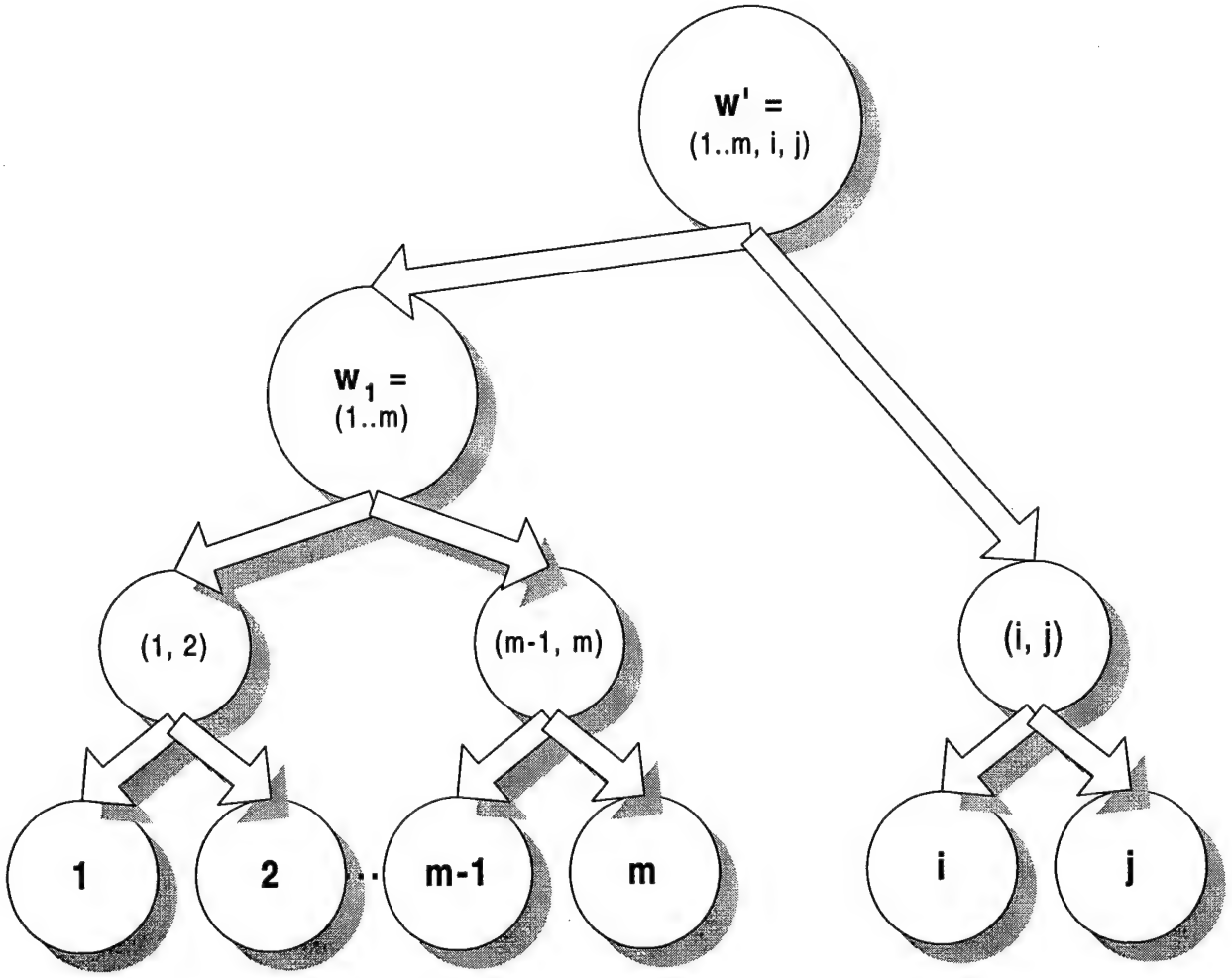


Figure 4.4 Combination of a Tree with an Independent Node Pair—A Tree-Pair Join.

■ Definition 8. The combination (join) of a tree w, $w = (1, \ldots, m)$, and a node pair (i, j) is a new tree $T_{w.ij} \mid$ w = value(T.left), (i, j) = value(T.right), and w' = value(root) = (1, .., m, i, j) and I will refer to this type of combination operation as a tree-pair join. Figure 4.4 illustrates this definition.

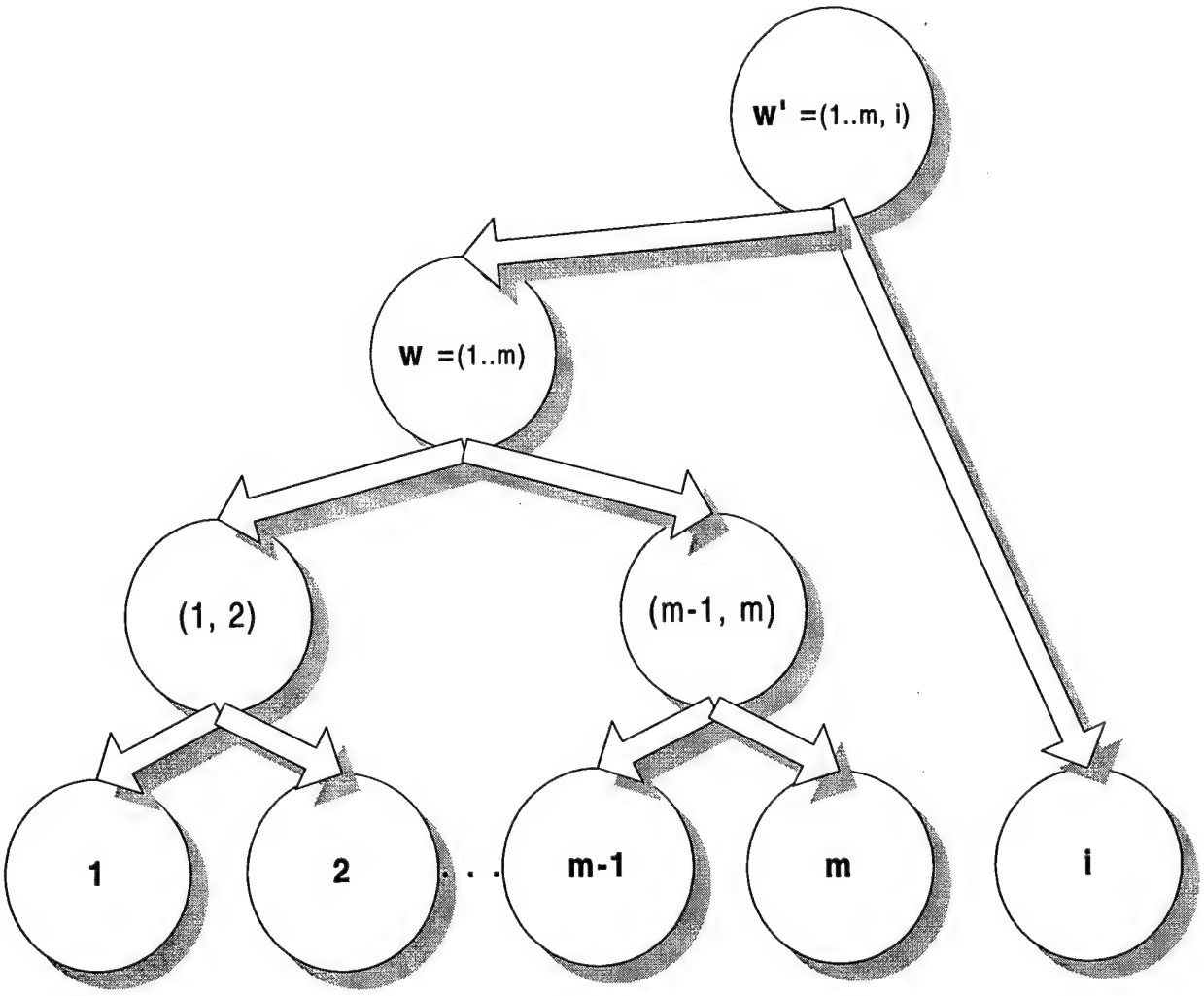


Figure 4.5 Combination of a Tree with a Single Leaf—A Tree-Leaf or Tree-Single Join.

- Definition 9. The combination (join) of a tree w, $w = (1, \ldots, m)$, and single leaf node i is a new tree $T_{w,i}$ | w = value(**T**.left), i = value(**T**.right), and w' = value(root) = (1, .., m, i), and I will refer to this type of combination operation as a tree-leaf or tree-single join. Figure 4.5 illustrates this definition.

The two reduction operations are to reduce the number of leaves in the leaf list, ***I***, which is simply set subtraction, and to reduce the number of quadruples stored in the quadruple heap, ***Q'***. If the quadruples are all stored in a single heap, then every time a particular leaf ***i*** is added to a tree, all quadruples in the heap which contain ***i*** should be removed.

- Definition 10. Leaf reduction is

$$I = I - i, i \in I.$$

- Definition 11. Quadruple heap reduction, where i, j, k, l refer to fields in the quadruple q and to leaves in I, is

$$Q' = Q' - q \mid q = \langle i\ j\ k\ l \rangle, i, j, k, l \in I.$$

4.2.4 Component Algorithms

My implementation of the METD2 algorithm consists of several constructing and deconstructing algorithms and data structures. These structures include the (1) leaf node quadruples heap, ***Q***; (2) quadruple heap bucket list, ***Q'***; (3) quadruple heap kill list, ***K***; (4) list of independent leaf nodes, ***I***; and (5) forest, ***F***, of trees, ***T***, constructed. The purpose of the heap bucket list ***Q'*** is to reduce the run-time for the algorithm. Since it contains exactly the same members as ***Q***, the algorithm can be correctly implemented on ***Q***, rather than on ***Q'***, although it will run slower. Likewise, the list of quadruples in ***Q*** or ***Q'*** to delete for a particular leaf node I_i is also designed to reduce the run-time of the algorithm. The naive approach to deleting quadruples in ***Q*** or ***Q'*** deletes quadruples whose component leaves are not in ***I*** when they are first encountered in the heap. This method also works correctly, although more slowly, and eliminates the need to use the heap kill list ***K***.

4.2.4.1 Build Quad Heap and Build Quad Buckets

The last step (Step 4) of the preprocessing stage (Steps 1-4) of the METD2 algorithm (Figure 2.1) constructs the initial quadruple heap Q, the list of heaps Q', and the list of kill heaps K. The algorithm for Q' essentially copies the contents of Q into the component heaps q_i in Q' (since Q does not have any component heaps, but instead is one continuous heap,, the notation q_i shall refer to q'_i in Q'. The procedure for K essentially copies the "left-over" elements not in each q_i into each k_i of K.

Figure 4.6 illustrates the algorithm to build the initial quadruple heap Q:

Do generate all $\binom{n}{4}$ possible combinations of four leaves (i, j, k, l) in I.

For each (i, j, k, l) combination **do** $q \leftarrow \min (d_{(i,j)\backslash(k,l)})$.

Do heap sort in ascending order all $\binom{n}{4}$ quadruples q in Q.

Figure 4.6 Algorithm for Constructing the Initial Quadruple Heap, Q.

Figure 3.6(a)-(c) show the three possible topological configurations. For Exact Tree Decomposition, all three equations must be true, which leads to the configuration of Figure 3.6(d). Since the data may contain errors, or the source distribution may not be perfectly tree-decomposable, our goal is to find the minimal error. The minimal error is simply the minimum of these three errors, as defined by Equation (10a) in Section 4.2.2. An alternate approach for calculating the minimum error is to find the average of these three errors, as defined by Equation (10b) of Section 4.2.2.

Figure 4.7 details the algorithm to build the lists of quadruple heaps and kill heaps, Q' and K:

For all q_i in Q' **do** $q_i \leftarrow \varnothing$.

For all k_i in K **do** $k_i \leftarrow \varnothing$.

For all q in Q $|q = <i\ j\ k\ l>$ **do**

$q_i \leftarrow q_i + <i\ j\ k\ l>$.

$k_j \leftarrow k_j + <i\ j\ k\ l>$.

$k_k \leftarrow k_k + <i\ j\ k\ l>$.

$k_l \leftarrow k_l + <i\ j\ k\ l>$.

Figure 4.7 Algorithm for Constructing the Quadruple Heaps Lists, Q' and K.

Each of the lists of quadruple heaps, Q' and K, has members, q_i and k_i corresponding to the indexes for the variables, i in I, in the database, D. Initially each component heap, q_i and k_i, has no elements. The **for** loop moves through the initial quadruple heap Q and copies each element to one component heap in Q', q_i, corresponding to the first field i in the quadruple <i j k l>, and also copies each element to three component heaps in K, k_j, k_k and k_l, corresponding to the second, third, and fourth fields j, k, l in the quadruple <i, j, k, l>.

4.2.4.2 Extract Minimum Cross-Correlation Error

METD2's main loop (Steps 5-10 of Figure 2.1) compares four type of possible topological joins (tree-tree, tree-single, tree-pair, quad). When constructing the hidden-valued tree, METD2 prefers, given topological joins with the same cross-correlation error, to build a new tree from a

quad first, from a tree-tree join second, from a tree-pair join third, and from a pair-pair join least. The first of these (Step 6 of Figure 2.1) is to find the minimum quadruple q in Q'.

Since the preprocessing stage resulted in a list of n-3 heaps, q_i in Q, we simply compare the minimum-error (the "first") quadruple in each heap, as shown in Figure 4.8:

$q_{min} \leftarrow q_o$

For all heaps q_i in $(Q' - q_o)$ **do**

if $q_i < q_{min}$ **then** $q_{min} = q_i$.

Figure 4.8 Algorithm for Finding the Minimum-Error Quadruple, q_{min}.

For Step 7, to determine the minimum-error tree-tree combination, we iterate over the forest, F, to select pairs of trees, Ti, and Tj from F. For each pair, we record the maximum error between those two trees. Then, for all pairs of trees in F, we record the minimum error (of the maximum errors we previously calculated), as shown in Figure 4.9:

$w_{min} \leftarrow d_{wi\backslash wj}$; $w_i, w_j \in F$.

For each of the $\binom{|F|}{2}$ combinations of trees in F do.

if $d_{wi\backslash wj} < w_{min}$ **then** $w_{min} \leftarrow \min d_{wi\backslash wj}$.

Figure 4.9 Algorithm for Finding the Minimum-Error Tree-Tree Join, w_{min}.

For Step 8, to determine the minimum-error tree-pair combination, we iterate over the forest, F, to select a tree, T, and over the set of leaf nodes, I, to select a pair of leaves, (i, j). For each pair of leaves (k, l) in T, we need to find the maximum error between them and the pair of

leaves (i, j) in I. Then, for all pairs of trees and leaves, we need to find the minimum error (of the maximum errors we previously calculated), as detailed in Figure 4.10:

$ijw_{min} \leftarrow d_{(i,j)\setminus w}$, $i, j \in I$, $w = T \in F$.

For each of the $\binom{|I|}{2}$ pairs of leaves (i, j) in I do.

For each tree $T = w$ in F do.

if $d_{(i,j)\setminus w} < ijw_{min}$ **then** $ijw_{min} \leftarrow d_{(i,j)\setminus w}$.

Figure 4.10 Algorithm for Finding the Minimum-Error Tree-Pair Join, ijw_{min}.

Finding the minimum-error tree-leaf combination (Step 9) follows a similar process as for Step 8 above, as illustrated in Figure 4.11:

$iw_{min} \leftarrow d_{i\setminus w}$, $i, \in I$, $w = T \in F$.

For each leaf i in I do.

For each tree $T = w$ in F do

if $d_{i\setminus w} < iw_{min}$ **then** $iw_{min} \leftarrow d_{i\setminus w}$).

Figure 4.11 Algorithm for Finding the Minimum-Error Tree-Single Join, iw_{min}.

4.2.4.3 Build Tree

After determining the minimum-cross-correlation-error structural join for each of the four types of structures, METD2 constructs a tree from the minimum-error combination among the four joins. Build tree simply makes a binary tree whose left child is one of the join objects and whose right pair is the other. Figure 4.12 illustrates this operation:

$T_{join} \leftarrow \min (q_{min}, w_{min}, ijw_{min}, iw_{min})$.

Figure 4.12 Algorithm for Building a Tree.

4.2.4.4 Remove Leaves and Quadruples

Step 10 of METD2 removes the leaves used to build a new structure (tree-tree, tree-pair, tree-single, quad) from the list of independent nodes, I, and also removes all quadruples, q, containing a member whose index corresponds to a leaf used to build a new structure. The definition for deleting leaves from I at the end of section 4.2.2 is sufficient for pseudo-code generation in and of itself. Likewise, tree deletion is also simply set removal. However, elimination for the kill-lists and the heap-lists is more involved.

Figure 4.13 shows the algorithm for removing quadruples q in the heaps q_i in $\boldsymbol{Q'}$:

For all i $\in T_{join}$ do $q_i \leftarrow \varnothing$.

For all j | j < greatest i | i $\in T_{join}$ do

While $k_j \neq \varnothing$ do.

$k_j \leftarrow k_j - q \mid q \in k_j$.

$q_m \leftarrow q_m - q \mid q = <$m n k l$>$, q$\in k_j$.

Figure 4.13 Algorithm for Removing Quads q from Heap Lists $\boldsymbol{K}$ and $\boldsymbol{Q'}$.

4.2.5 Parameter Estimation

The first phase of parameter construction, where we determine the correlation of the hidden nodes, generates equations and $2(n-1)$ unknown expressions. Thus, as long as $n \geq 4$, the matrix method described in this section will generate a unique solution or no solution.

In the second phase of parameter construction, METD2 determines the prior probabilities of the hidden nodes $P(w)$, and the conditional probabilities of the hidden nodes given the

observable nodes $P(w|\, x)$ and given another hidden node $P(w_i|\, w_j)$. Since the algorithm assumes that the hidden variable "causes" the observed variables in D, METD2 completed the arc direction step at the same time it assigns arcs (in the tree construction phase). This assumption also permits the derivation of several equations from the casual structure of the belief network constructed in the previous phase.

The method suggested by [Liu et al, 1990] is a log-linear, matrix, linear-programming approach. Although Liu does not does not explain the equations behind his approach, we see that the parameter determination process should have two phases. The first phase consists of finding the correlation coefficients for the hidden variables and the second phase consists of finding the conditional probabilities on the hidden variables. For the first phase of the process there are $n(n-1)$ equations and $2(n-1)$ unknowns, and for the second phase of the process there are $n(n-1)$ equations and $3(n-1)$ unknowns. Each stage of the process has more equations than unknowns so that the equations listed by Liu are over-determined: there is either a unique solution or an approximate solution.

4.3 Computational Complexity

The complexity of the algorithm, as a whole, is directly a combinatorial function that results from processing the n observable variables in the source sample D. Since the procedure must consider all possible combinations of four leaves (quadruples), there are $\binom{n}{4}$ unique quadruples to consider, and the complexity of the algorithm has to be some function of this combinatorial, i.e.,

$O\left(f\binom{n}{4}\right)$. Since the algorithm adds $n-1$ hidden nodes w_i to the tree, the function $f(x)$ must be similar to $f(x) = cnx$, where c is some constant. The asymptotic complexity of the algorithm should be $O\left(n\binom{n}{4}\right) = O(n(n^4)) = O(n^5)$. Detailed analysis reveals that, indeed, the worst case asymptotic bound for the algorithm is $O(N^5)$, while its best case bound is $O(N^4)$. The experiments conducted (described in Chapter 5) revealed that the typical run-time behavior of the algorithm is its best case—$O(N^4)$.

4.3.1 Preprocessing Steps

For the preprocessing stages, Steps 1 through 4 of the algorithm in Figure 2.1, the computational complexity is relatively straight-forward, although determining the complexity of the two lists of sorted quadruple buckets, Q' and K, is rather involved. Note that the algorithm's "big O" notation is a function of the number of *variables* in each sample in the database as opposed to the number of samples. Thus, ignoring the preprocessing stage of determining the correlation on the variables (Step 3 in Figure 2.1), the algorithm will run the same speed for a database with 1000 cases as for a database with 1,000,000 cases, as long as each database has the same number of variables and the same underlying domain model. The following analysis shows that the tight bound for the complexity of the preprocessing steps is

$$O(N^4) \leq \text{time} \leq O(N^4 lgN).$$

4.3.1.1 Constructing the Leaf list and the Correlation Matrix

The time to construct the list of independent leaf nodes ***I***, where n is the number of variables in the database ***D***, is

$$O(N).$$

The time to construct the correlation matrix on the variables in ***D***, where m is the number of samples and n is the number of variables per sample, is

$$O(mn^2) = \begin{cases} O(N^2), m << n; \\ O(N^3), m \geq n. \end{cases}.$$

4.3.1.2 Constructing the Lists of Quad Heaps

Given an *NlgN* sorting algorithm, such as heap sort, the time to construct the sorted quadruple heap, ***Q'***, where n is the number of variables in ***D*** is

$$O\left(\binom{n}{4}\log_2\binom{n}{4}\right) = O\left(\left(\frac{1}{24}n(n-1)(n-2)(n-3)\right)\log_2\left(\frac{1}{24}n(n-1)(n-2)(n-3)\right)\right) \quad (11)$$

$$= O\left(\mathrm{N}^4\log_2\left(\mathrm{N}^4\right)\right) = O\left(\mathrm{N}^4\log_2 N\right).$$

The time to construct the heap list of quadruple buckets, ***Q'***, where n is the number of variables in ***D***, and Q'_i refers to the quadruple bucket for the variable in ***D*** labeled as i, is

$$O\left(\sum_i \left|Q'_i\right|\right), \qquad \text{where } \left|Q'_i\right| = \binom{n-i-1}{3}, i = \{0,1,2,\cdots,n-4\},$$

$$= O\left(\binom{n}{4}\right) = O\left(\frac{1}{24}n(n-1)(n-2)(n-3)\right) = O\left(\mathrm{N}^4\right) \qquad (12).$$

Each of the n "kill" lists K_i contains only items which were not also entered in the corresponding heap list, Q'_i. The time to construct the kill list of quadruple buckets, K, where n is the number of variables in D, is

$$O\left(\sum_i |K_i|\right), \quad \text{where} \quad |K_i| = \binom{n-1}{3} - Q'_i, i = \{0,1,2,\cdots,n\}, \text{ and } |Q'_n| = |Q'_{n-1}| = |Q'_{n-2}| = |Q'_{n-3}| = 0,$$

$$= O\left(n\binom{n-1}{3} - \binom{n}{4}\right) = O\left(\frac{1}{8}n(n-1)(n-2)(n-3)\right) = O(\mathrm{N}^4) \qquad (13).$$

4.3.2 Topology Construction Phase

Within the main loop, there are four combination operations—(leaf pair, leaf pair), (tree, single leaf), (tree, leaf pair), (tree, tree)—and two deletion operations—remove(leaf) and remove(quadruple). Since the number of each of these operations depends upon each of the other operations, the complexity analysis becomes somewhat complicated for the (tree, tree) and (tree, leaf) combinations. The following analysis shows that the tight bounds for the complexity of the topology construction phase is

$$O(N^4) \leq \text{time} \leq O(N^5).$$

4.3.2.1 Complexity of the Tree-Single Join

The best case time for the tree-leaf combination occurs in the sequence where each pass through the main loop of the algorithm removes a new quadruple from Q' and adds a tree T to the forest of trees, until no leaves remain in I. Thus, there are k trees in the forest F and n-$4k$ leaves in I. Each increment of k corresponds to a pass through the main loop of the METD2

algorithm and removing four leaves from $\boldsymbol{I}$ and adding one tree to $\boldsymbol{F}$. Each tree in $\boldsymbol{F}$ has four leaves and the cross-correlation error metric requires that we pick three leaves from the tree:

$$\binom{4}{3}\left(\left(\sum_{k=1}^{\left\lfloor\frac{n}{4}\right\rfloor-1} k(n-4k)\right)+(n \bmod 4)\left\lfloor\frac{n}{4}\right\rfloor\right) \tag{14}$$

$$= O\left(\sum_{k=1}^{\left\lfloor\frac{n}{4}\right\rfloor-1} k(n-4k)\right) = O\left(\sum_{k=1}^{\left\lfloor\frac{n}{4}\right\rfloor-1} kn - \sum_{k=1}^{\left\lfloor\frac{n}{4}\right\rfloor-1} 4k^2\right) = O\left(\frac{n^2(n+1)}{2} + \frac{k(k+1)(2k+1)}{6}\right)$$

$$= O(N^3).$$

The worst case time for the tree-leaf combination occurs in the sequence where, after the initial tree is created, only a singly leaf is added to the tree at a time. Thus, on pass k there is one tree and n-k leaves remaining, but the one tree has k leaves. Again, the cross-correlation error metric requires that we pick three of the k leaves from the single tree in $\boldsymbol{F}$:

$$\sum_{k=4}^{n-1}(n-k)\binom{k}{3} \tag{15}$$

$$= O\left(\sum_{k=1}^{n-4}(n-k-3)\binom{k+3}{3}\right) = O\left(\sum_{k=1}^{n-1}(n-k)\frac{(k+3)(k+2)(k+1)}{6}\right) = O\left(\sum_{k=1}^{n-1} nk^3\right)$$

$$O\left(\frac{n^3(n+1)^2}{4}\right) = O(N^5).$$

Both the worst-case and best-case tree-leaf join scenarios assume that there is no remainder, i.e., that $n \bmod 4 = 0$. Later complexity computations either explicitly or implicitly handle remainder leaves. Thus the tight bound for the computational complexity of a tree-leaf join is

$$O(N^3) \leq \text{time} \leq O(N^5).$$

4.3.2.2 Complexity of the Tree-Pair Join

The best case time for the tree-leaf pair combination occurs in the sequence where each pass through the main loop of the algorithm removes a new quadruple from ***Q'*** and adds a tree ***T*** to the forest of trees, until no leaves remain in ***I***. If there is a remainder, i.e. *n mod 4* $\neq 0$, then we add some constant number of calculations:

$$\left(\binom{4}{2}\sum_{k=1}^{\left\lfloor\frac{n}{4}\right\rfloor-1} k\binom{n-4k}{2}\right)+\begin{cases}\binom{4}{3}\left\lfloor\frac{n}{4}\right\rfloor, n \bmod 4 = 1\\ \binom{4}{2}\left\lfloor\frac{n}{4}\right\rfloor, n \bmod 4 = 2\\ \binom{4}{3}\left\lfloor\frac{n}{4}\right\rfloor+\binom{4}{2}\left\lfloor\frac{n}{4}\right\rfloor+\binom{4}{3}\left(\left\lfloor\frac{n}{4}\right\rfloor-1\right)+\binom{6}{3}, n \bmod 4 = 3\end{cases} \quad (16)$$

$$= O\left(\binom{4}{2}\sum_{k=1}^{\left\lfloor\frac{n}{4}\right\rfloor-1} k\binom{n-4k}{2}\right) = O\left(6\sum_{k=1}^{\left\lfloor\frac{n}{4}\right\rfloor-1} k\frac{(n-4k)(n-4k-1)}{2}\right) = O\left(3\sum_{k=1}^{\left\lfloor\frac{n}{4}\right\rfloor-1} k(n-4k)(n-4k-1)\right)$$

$$= O\left(\sum_{k=1}^{\left\lfloor\frac{n}{4}\right\rfloor-1} kn^2+k^2\right) = O\left(\frac{n^3(n+1)}{3}+\frac{n(n+1)(2n+1)}{6}\right)$$

$$= O(N^4).$$

The worst case time for the tree-leaf pair combination occurs in the sequence where, after the initial tree is created, only a pair of leaves is added to the tree at a time. In this situation, the single tree in the forest has *2(i+1)* leaves and there are *n - 2(i + 1)* leaves remaining in ***I***. We pick two leaves from ***I*** and two leaves from the single tree in ***F***. Equation 17 neglects the constant number of computations necessary to handle any remainder of leaves, i.e. *n mod 4* $\neq 0$.

$$\binom{4}{2}\sum_{k=1}^{\left\lfloor\frac{n-4}{2}\right\rfloor-1}\binom{2(k+1)}{2}\binom{n-2(k+1)}{2} \qquad (17).$$

$$= O\left(6\sum_{k=1}^{\left\lfloor\frac{n-4}{2}\right\rfloor-1}(k+1)k\frac{(n-2(k+1))(n-2k+1)}{2}\right) = O\left(\sum_{k=1}^{\left\lfloor\frac{n-4}{2}\right\rfloor-1}k^4\right) = O\left(\frac{n^5}{5}+\frac{n^4}{2}+\frac{n^3}{3}-\frac{n^1}{30}\right)$$

$$= O(N^5).$$

Therefore, the tight bound for the computational complexity of the tree-pair join is

$$O(N^4) \le \text{time} \le O(N^5).$$

4.3.2.3 Complexity of the Tree-Tree Join

The complexity analysis for the tree-tree combination is rather complex. The algorithm adds trees to ***F*** as it removes leaves from ***I*** and quadruples from ***Q'***. Then, the algorithm reduces the trees in ***F*** until there is only a single tree. The best-case scenario for building and reducing the forest of trees ***F*** is when METD2 adds a new tree to ***F*** on each pass through the main loop of Figure 2.1. In this scenario, the algorithm only adds trees with less than or equal to the number of leaves to any tree in the rest of the forest, and only when all trees have the same number of leaves, adding a tree which has more leaves than its predecessors. Thus, at the start of the $\left\lfloor\frac{n}{4}\right\rfloor+1$ pass, there are $\left\lfloor\frac{n}{4}\right\rfloor$ trees, and each tree has four leaves. Equation 18 neglects the constant number of computations necessary to handle any remainder of leaves, i.e. $n \bmod 4 \neq 0$:

$$\binom{4}{2}^2\sum_{k=1}^{\left\lfloor\frac{n}{4}\right\rfloor-1}\binom{k+1}{2} \qquad (18).$$

$$= O\left(36\sum_{k=1}^{\left\lfloor\frac{n}{4}\right\rfloor-1}\frac{(k+1)k}{2}\right) = O\left(18\sum_{k=1}^{\left\lfloor\frac{n}{4}\right\rfloor-1}(k^2+k)\right) = O\left(\frac{n(n+1)(2n+1)}{6}\right) = O(N^3).$$

The equation above simply chooses two leaves from each of two trees, each of which has four leaves. On each pass, there are *k+1* trees in ***F*** and the algorithm examines all pairs of such trees.

This sum is added to the cost of reducing the forest ***F*** to one element. In the best case scenario, each pass through the main METD2 main loop removes one tree from the forest, so that $\left\lfloor\frac{n}{4}\right\rfloor-1$ passes are necessary to reduce ***F*** to one tree. Each pass through the forest only removes trees with less than or equal to the number of leaves of all existing trees, and any new tree created cannot have more than twice as many leaves as any other tree. If we count the remainder, the precise calculation for the number of passes necessary is $2\left(\left\lfloor\frac{n}{4}\right\rfloor-1\right)+rem\left\lfloor\frac{n}{4}\right\rfloor$ in order to create the final tree. Ignoring the remainder, this leads to Equation 19:

$$\sum_{i=1}^{\log_2\left\lfloor\frac{n}{4}\right\rfloor}\sum_{j=1}^{\left\lfloor\frac{n}{2^{i+2}}\right\rfloor-1}\binom{2^i+1}{2}^2\left(\left\lfloor\frac{n}{2^i+2}\right\rfloor-j+1\right)+\binom{2^i+2}{2}^2\binom{j+1}{2}+j\left(\left\lfloor\frac{n}{2^{i+1}}\right\rfloor-j\right)\binom{2^i+1}{2}\binom{2^i+2}{2} \quad (19).$$

The two summations, together, lead to *n-1* evaluations of the complex inner term. The inner term consists of three simpler terms. The first of the inner terms refers to choosing from the forest ***F*** two trees which both have 2^{i+2} leaves, and the second inner term refers to choosing from ***F*** two trees which both have 2^{i+1} leaves. The previously stated assumptions for the best-case scenario lead to the observation that each time the inner summation completes, there are ½ as many trees as in the previous summation, and each tree has twice as many leaves as the typical

tree of the previous iteration. Thus, the $\left\lfloor \frac{n}{2^{i+1}} \right\rfloor$ sub-term describes the number of trees in the forest which have 2^{i+1} leaves at the beginning of the pass, and j gives the number of trees with 2^{i+2} leaves at the beginning of the pass. The third of the three inner terms calculates the number of comparisons between trees with 2^{i+1} leaves and 2^{i+2} leaves on a given pass. Also note, Equation 19 neglects the constant number of computations necessary to handle any remainder of leaves, i.e. $n \bmod 4 \neq 0$. Evaluating Equation 19 leads to the best-case forest $\boldsymbol{F}$ reduction of

$$O(N^4).$$

The worst-case computational complexity for the tree-tree join occurs in the scenario where METD2 creates two trees in $\boldsymbol{F}$ after the first two passes and creates more trees. At the end of the $\left\lfloor \frac{n}{4} \right\rfloor + 1$ pass, there are exactly two trees. Then, the reduction of two trees to one tree occurs inconstant time, and the total tree-tree join's computational time is driven by the build-up phase for $\boldsymbol{F}$. This leads to Equation 20:

$$\sum_{k=1}^{\left\lfloor \frac{n-8}{2} \right\rfloor - 1} \binom{3+k+1}{2}\binom{3+k}{2} + \sum_{k=1}^{\left\lfloor \frac{n-8}{2} \right\rfloor} \binom{3+k}{2}\binom{3+k}{2} \qquad (20).$$

Equation 20 looks similar to Equation 18—evaluating the expression in Equation 20 yields, as does the expression in Equation 18, a worst-case forest $\boldsymbol{F}$ construction of

$$O(N^5).$$

There are n leaves in $\boldsymbol{I}$. Each of the two trees initially removes 4(2) = 8 leaves from $\boldsymbol{I}$, so that there are n-8 leaves remaining. Each pass through the main loop of METD2 removes exactly one more leaf from $\boldsymbol{I}$ and on even passes, one tree has one more leaf than the other, while on odd

passes they have the same number of leaves. The equation above simply chooses two leaves from the tree with $3+k$ leaves on the k^{th} pass (grouping all the even passes together) and two leaves from the tree with $3+k+1$ leaves on the $k^{th}+1$ pass. Also note, Equation 20 neglects the constant number of computations necessary to handle any remainder of leaves, i.e. $n \bmod 4 \neq 0$.

As a result of analyzing the best and worst cases for the two phases of constructing the forest of trees, $\boldsymbol{F}$, and reducing the forest of trees, we see that the tight computational complexity bounds for the tree-tree join are the same as for the tree-pair join:

$$O(N^4) \leq \text{time} \leq O(N^5).$$

4.3.2.4 Complexity of the Leaf and Quadruple Heap Reductions

Leaf deletion is $O(N^2)$. Deleting nodes from the quadruple heap $\boldsymbol{Q'}$ is

$$= O\left(\binom{n}{4}\right) = O\left(\frac{1}{24}n(n-1)(n-2)(n-3)\right) = O\left(\mathrm{N}^4\right).$$

Deleting the quadruple bucket kill list $\boldsymbol{K}$ has the same complexity as constructing the list, $O(N^4)$, and the equation to derive the exact computational complexity is identical to the one used to derive the cost of constructing $\boldsymbol{K}$. However, implementing $\boldsymbol{K}$ as an unbounded memory structure, such as a linked list, will lead to much worse computational complexity. If each deletion in $\boldsymbol{Q'}$ requires searching from the beginning to the end of the list, and the length of each list is $O(N^3)$, then the bounds for the complexity lie between

$$O(N^6) \leq \text{time} \leq O(N^7).$$

Clearly, my METD2 implementation did not implement lists in this fashion.

4.3.2.5 Conclusion of Complexity Analysis for Topology Construction

The complexity for the topology construction phase is the sum of the complexities for each of the component stages, assuming either best-case (for the lower bound) or worst-case (for the upper-bound) performance. Fortunately, profiling the software reveals that the typical behavior of the algorithm is its best-case. Thus, the expected run-time behavior of the topology construction phase is

$$O(N^4) \leq \text{time} \leq O(N^5).$$

4.3.3 Parameter Determination Phase

The parameter determination phase determines the correlation on the hidden variables, i.e. p_{iw} and $p_{wi,wj}$. To determine the complexity of the parameter determination phase, we note that the size of each of the matrices is determined by the number of equations which can be derived from the topology of the marginalized belief network. For this analysis, we assume that the matrix $\boldsymbol{M} = \boldsymbol{A}^T \boldsymbol{A}$ is nonsingular, that is, that $\boldsymbol{M}$ has a unique inverse, $\boldsymbol{M}^{-1}$. The following analysis shows that the tight bounds for the complexity of the parameter determination phase is

$$O(N^5) \leq \text{time} \leq O(N^5).$$

The time to construct the log of weights correlation matrix, $\boldsymbol{X_w}$, is

$$O(2(n-1)) = O(N).$$

The time to construct the coefficient matrix, $\boldsymbol{A}$, is

$$O(2(n-1)) = O(N).$$

The time to construct the log of leaves correlation matrix, $\boldsymbol{X_i}$, is

$$O(n(n-1)^2) = O(N^3).$$

The time to construct the transpose coefficient matrix, $\boldsymbol{A}^T$, is

$$O(n(n-1)^2) = O(N^3).$$

The time to construct the square matrix $\boldsymbol{M} = \boldsymbol{A}^T\boldsymbol{A}$, is

$$O\left(2(n-1)\times\frac{1}{2}n(n-1)\times 2(n-1)\right) = O\left(2n(n-1)^3\right) = O\left(N^4\right).$$

The time to construct the inverse square matrix $\boldsymbol{M}^{-1} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}$, is

$$O(4(n-1)^2) = O(N^2).$$

The time to construct the regularized matrix $\boldsymbol{R} = \boldsymbol{M}^{-1}\boldsymbol{A}^T$, is

$$O\left(2(n-1)\times 2(n-1)\times\frac{1}{2}n(n-1)\right) = O\left(2n(n-1)^3\right) = O\left(N^4\right).$$

The time to construct the final matrix $\boldsymbol{X}_w = \boldsymbol{R}\,\boldsymbol{X}_i$ is

$$O\left(2(n-1)\times\frac{1}{2}n(n-1)\times 1\right) = O\left(n(n-1)^2\right) = O\left(N^3\right).$$

Thus the most time consuming operation in the first part of parameter determination is constructing the square matrix $\boldsymbol{M}$ and the regularized matrix $\boldsymbol{R}$.

The second half of the parameter determination process, which determines the prior probabilities $\boldsymbol{P(w)}$, and the conditional probabilities $\boldsymbol{P(w|\,x)}$ and $\boldsymbol{P(w_i|\,w_j)}$ is analyzed in later research.

5. EXPERIMENTAL RESULTS

The previous chapters provide an introduction to Bayesian domain models (Chapter 1), the METD2 algorithm (Chapter 2), previous and related research (Chapter 3), and the specific implementation details for METD2 (Chapter 4). In this chapter, I examine the results from running METD2 on data generated from three artificial problem domains I created and one real-world domain commonly used to benchmark learning algorithms. The first part of this chapter describes the data generation process and the actual experimental results from applying METD2 to the three artificial problem domains. The second part of this chapter details the results from applying METD2 to the one real world domain. The testing objective for both the artificial and real problem domains was to determine how well the cross-correlation metric could reproduce a tree-structured domain, with and without noise. I wished to determine the qualitative performance of the of Tree Decomposition. The third section of this chapter considers the asymptotic convergence of METD2. The final section of this chapter considers storage, space, and time when running the METD2 algorithm. Chapter 6 provides a discussion of these results and concludes this thesis.

5.1 Performance on Three Artificial Domains

For each test of METD2 on an artificial domain, I generated data from a source Bayesian domain model using stochastic simulation. The ***Ideal*** software package provided a basic framework for performing stochastic simulation. Since this thesis has focused on probabilistic networks and learning, the only type of nodes used in the construction process were what *Ideal* defines as *:chance, :prob* nodes. Each node in the tree was binary-valued, with a randomly-determined probability of being *true*. The sum of all probabilities for any node was 1. The second set of data (with noise) was created using *Ideal*'s *noisy-or-node-p* functions.

5.1.1 The Artificial Domains

I used three canonical tree-structures for testing METD2. The first structure is a balanced-binary tree with eight leaves and seven interior nodes (Figure 5.1). The second structure is an unbalanced-binary tree with seven leaves and six interior nodes (Figure 5.2). The third structure is an unbalanced-trinary tree with six leaves and four interior nodes (Figure 5.3). The following figures illustrate the constructed networks. From each of the three structures, I generated 10,000 noisy cases and ran METD2 on the database constructed using these cases. Since each of the artificial domains had eight or fewer variables, METD only had to complete three or, at most, four passes to generate a tree-structure. The running times for each domain was between 20 and 30 seconds on a dedicated Heward Parckard 715 with 64 MB of RAM, running C++ code under a HPUNIX operating system. I randomly generated the prior probabilities and the conditional probabilities for each domain network.

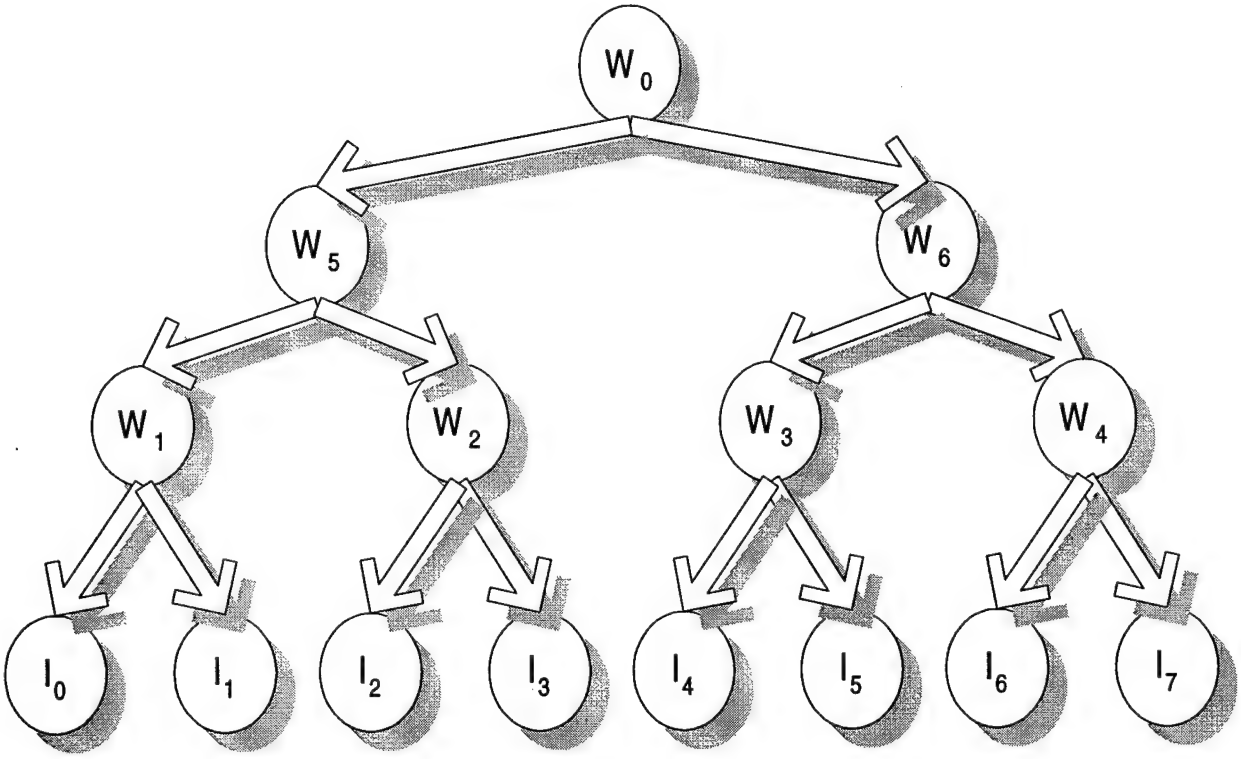


Figure 5.1 Balanced-Binary Tree Data Generator.

The balanced binary tree domain model of Figure 5.1 has fifteen nodes—eight observable (leaf nodes) and seven hidden (interior nodes). The interior nodes ($W_0 \ldots W_6$) marginalize the distribution over the domain variables ($I_0 \ldots I_7$). Each node has noisy probability of being on given that its parent is on.

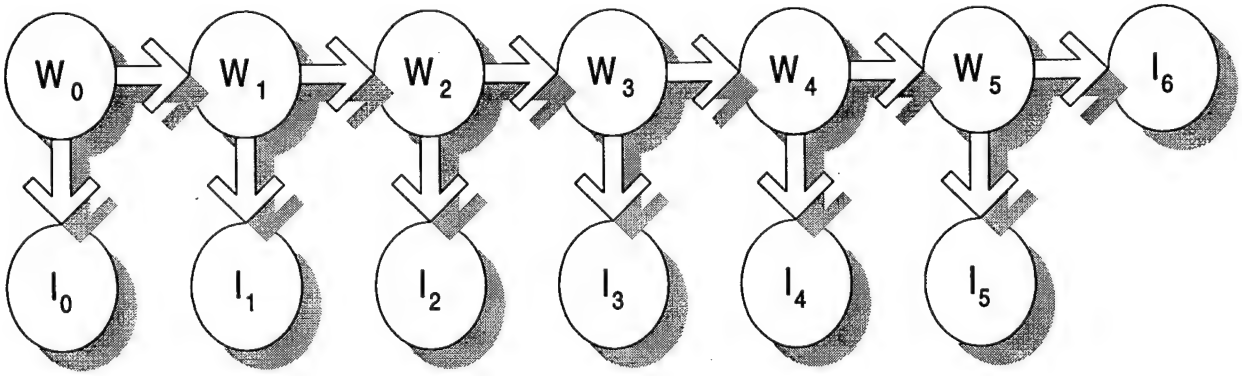


Figure 5.2 Unbalanced Binary Tree Data Generator.

The balanced binary tree domain model of Figure 5.2 has thirteen nodes—seven observable (leaf nodes) and six hidden (interior nodes). The interior nodes (W_0 . . . W_5) marginalize the distribution over the domain variables (I_0 . . . I_6). Each node has noisy probability of being on given that its parent is on.

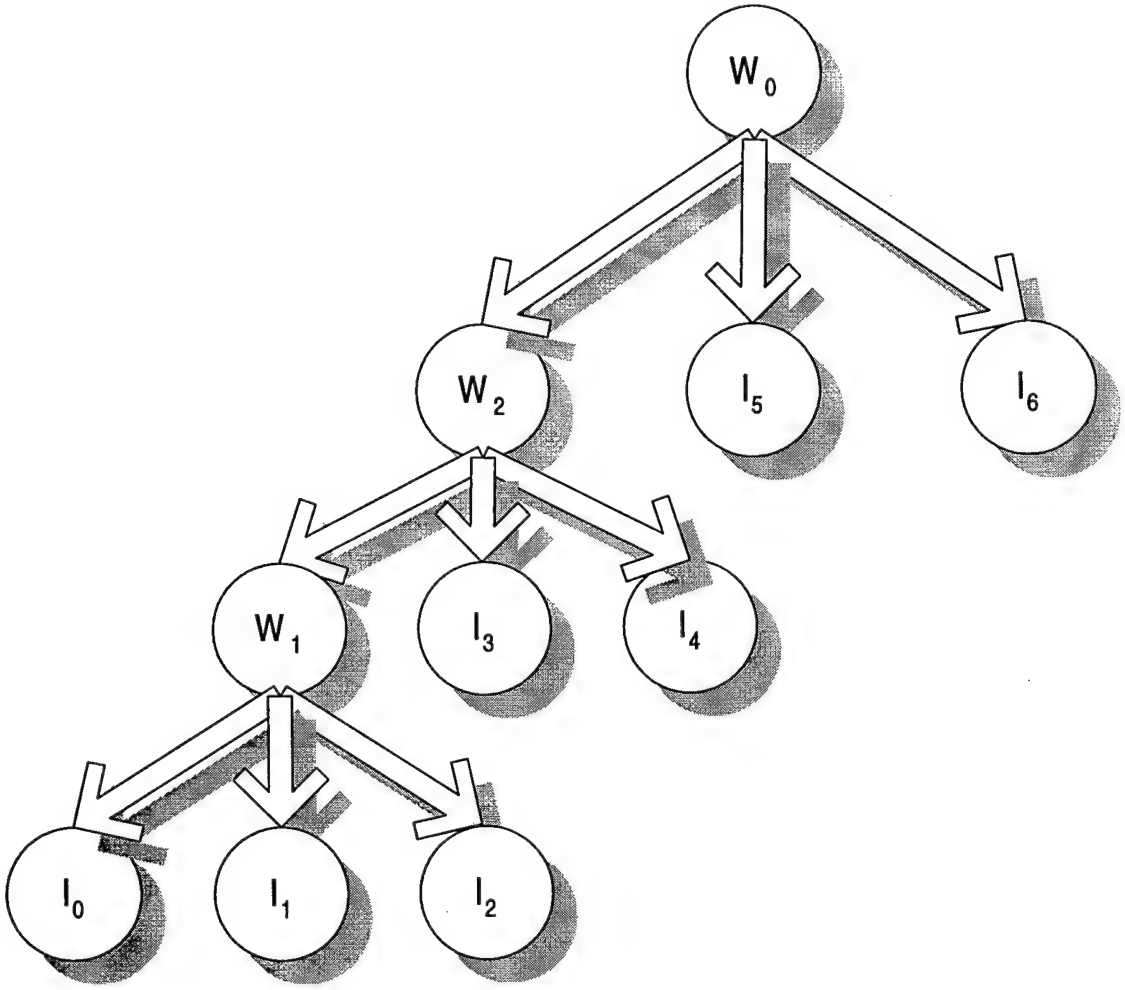


Figure 5.3 Unbalanced-Trinary Tree Data Generator.

The balanced binary tree domain model of Figure 5.3 has ten nodes—seven observable (leaf nodes) and three hidden (interior nodes). The interior nodes (W_0 . . . W_2) marginalize the distribution over the domain variables (I_0 . . . I_6). Each node has noisy probability of being on given that its parent is on.

5.1.2 Relationships Found on the Artificial Problem Domains

METD2 reproduced the balanced-binary tree-structured domain with a similar, but not identical, topological configuration, as illustrated in Figure 5.4. METD2's approximation of the unbalanced- binary tree-structured domain, as shown in Figure 5.5, with the closest balanced tree structure, had some surprising combinations. The third artificial source domain model, the unbalanced trinary tree, led to a completely different-looking topology, as shown in Figure 5.6.

Overall, these results were somewhat surprising. For example, in the case of the balanced-binary tree, I expected METD2 to recreate the original structure, since the algorithm assumes a tree-dependent distribution and this is what its inductive bias is designed to create. However, after observing the experiment, I recognized that the METD2 algorithm actually only groups nodes which have a low cross-correlation error. Therefore, if a parent's sibling had a low probability, METD2 might join the parent's sibling and one or more of its children in a structure, even if they were not siblings in the original architecture. The algorithm actually found an equivalent tree structure for the given data, but not necessary a structurally identical one. Also, the algorithm's inductive bias favors unbalanced trees over balanced trees, as demonstrated by the experimental results. It is not clear as of publication why the algorithm's bias was not for creating balanced trees instead.

One interpretation of these results is that the algorithm "juggles" the conditional probability tables, as the number of cases increases, to find the best tree-structure fit for the data. Thus, if a chain of high probability nodes behave similarly to another chain of nodes in a different part of the true (source) domain model, those nodes get grouped together. In a sense,

the algorithm "compacts" the resulting joint distribution. Although the resulting tree, viewed graphically, is clearly less "compact," the fusion and propagation of new information, mathematically, depends only upon a node's single parent, whereas in the original domain model, it would have possibly depended on multiple parents. Thus, the tree is more compact in the sense that it is more computationally efficient. This occurs because METD2's inductive bias causes the "fan-in" for each node to be either one or zero (in the case of the root node).

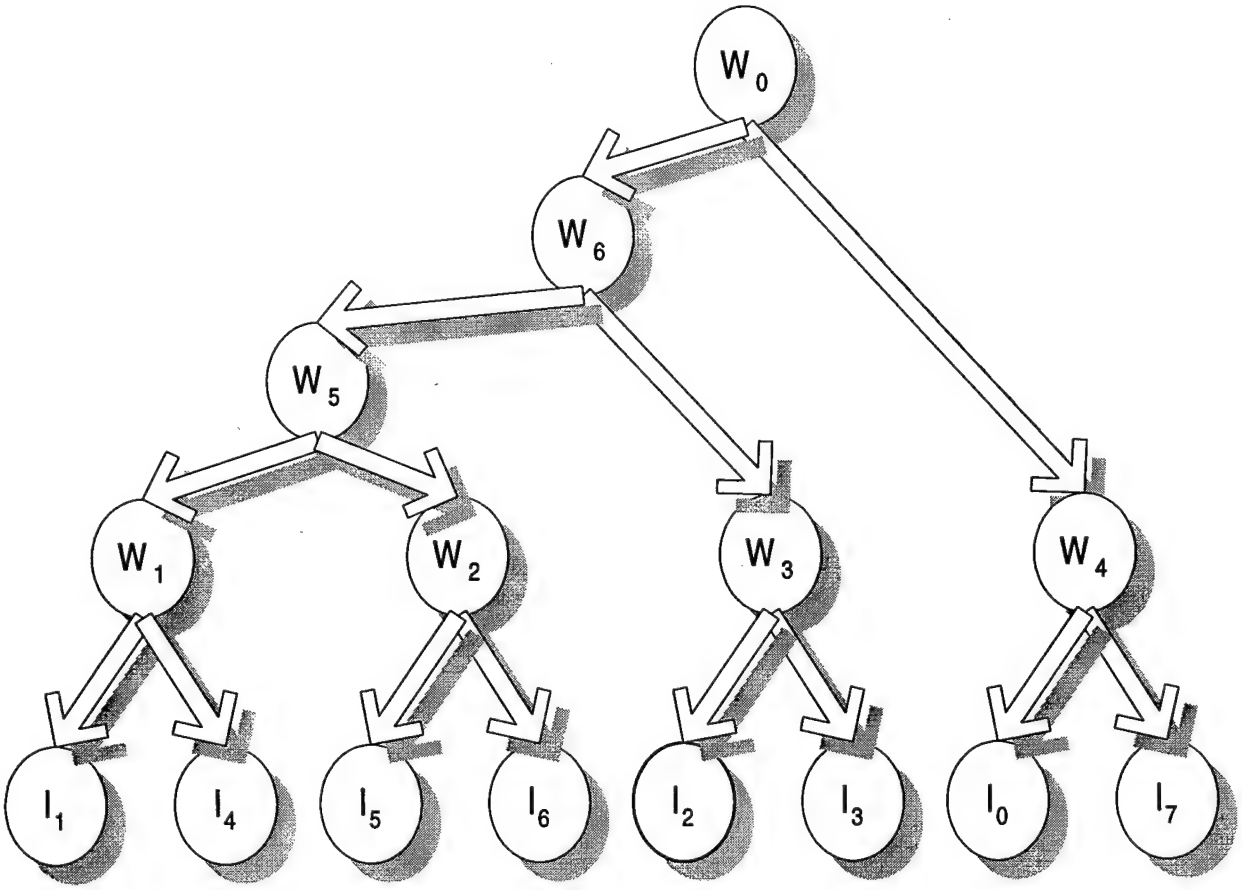


Figure 5.4 Balanced Binary METD2 Construction.

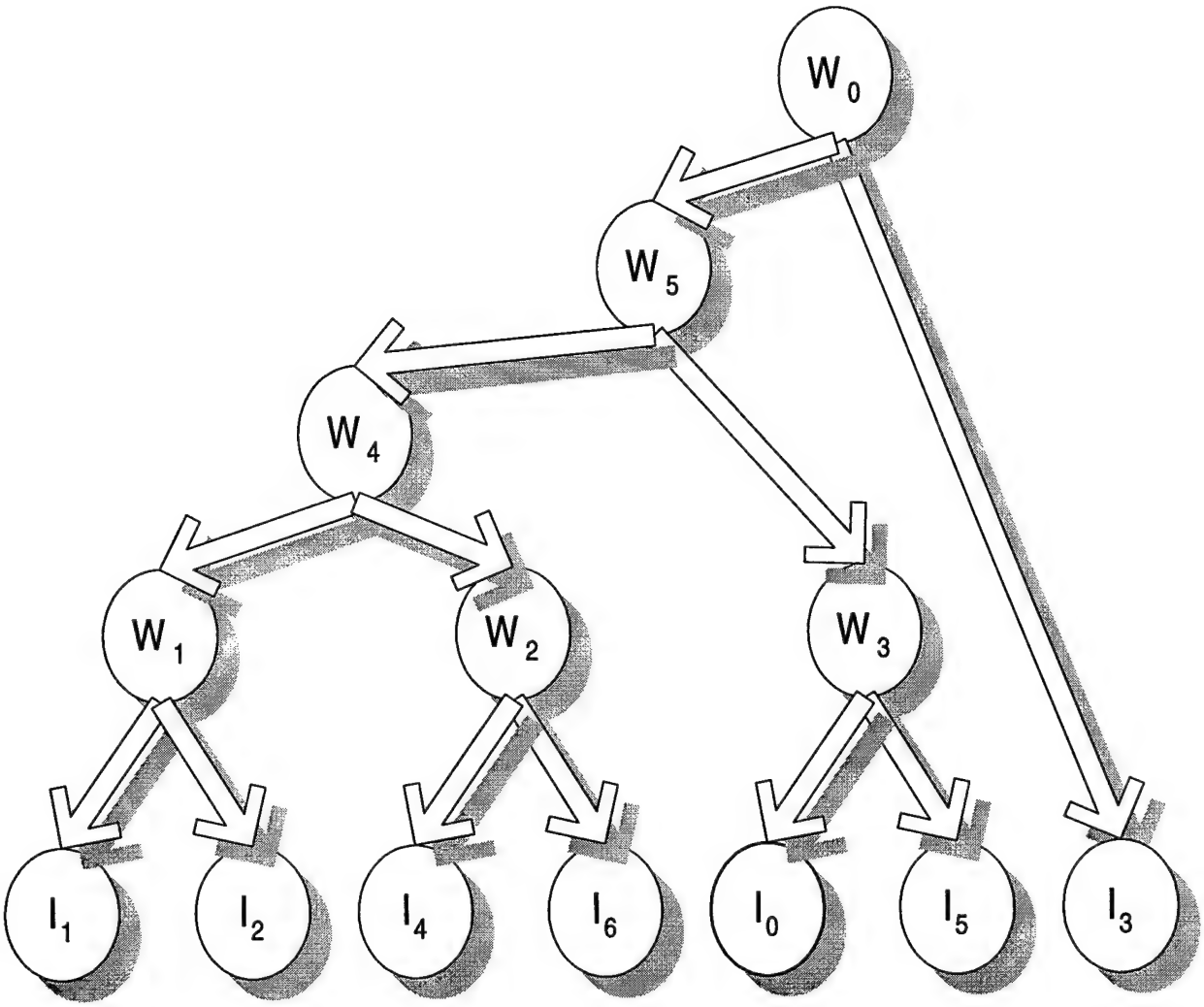


Figure 5.5 Unbalanced Binary Tree METD2 Construction.

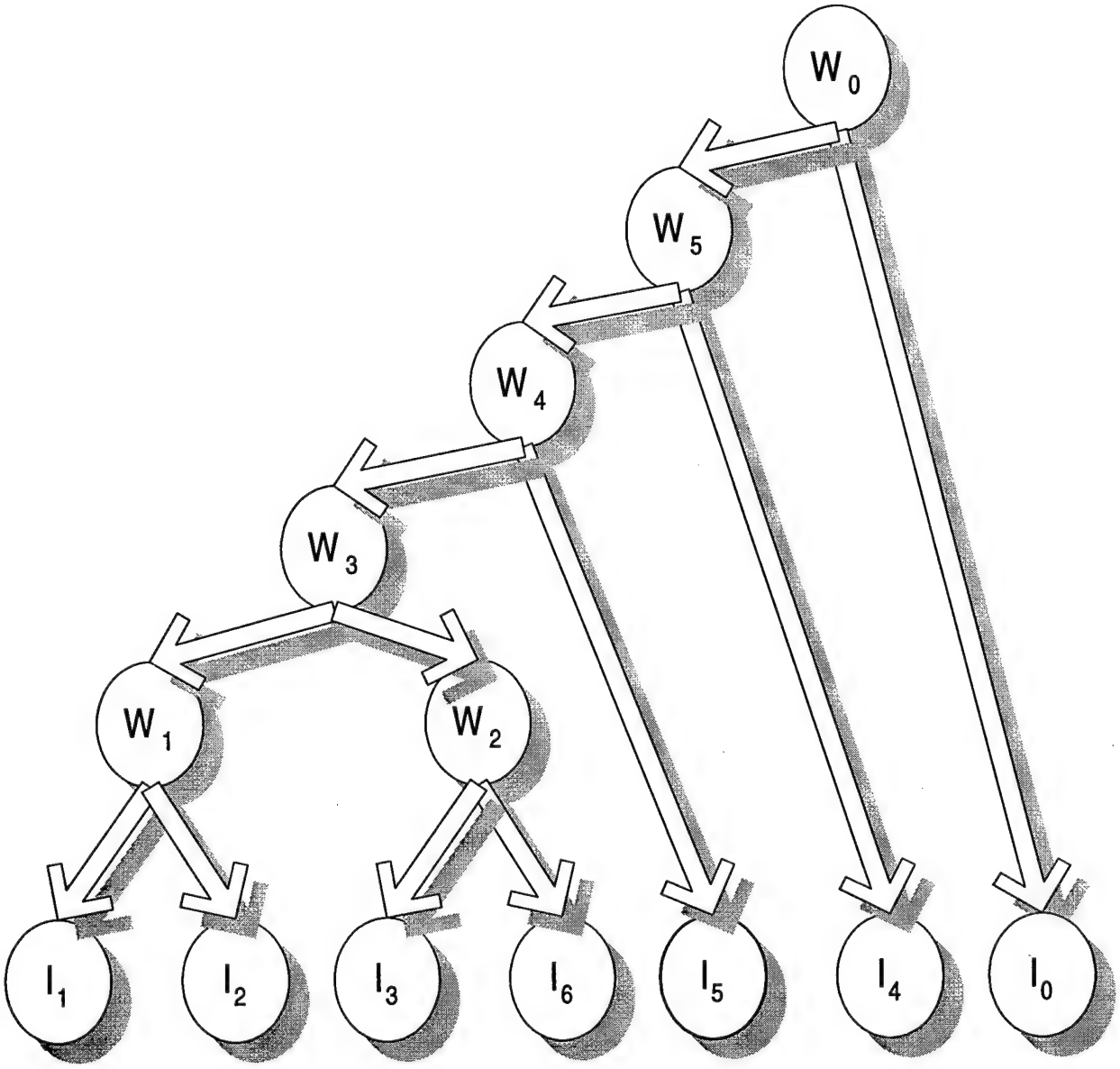


Figure 5.6 Unbalanced-Trinary Tree METD2 Construction.

5.2 Performance on a Real-World Domain

5.2.1 The Alarm Database

The objective of testing the algorithm with real-world domains was to determine if any of the cross-correlation relationships found by the algorithm corresponded to relationships found by other Bayesian applications. The benchmark domain chosen was the biomedical domain known as *Alarm*—based on the real-world domain of intensive-care-unit ventilator management, as described by [Sprites and Meek, 1995]. The data used was originally generated by [Herskovits, 1991] for testing Kutato/K2. Herskovits generated the data of 10,000 cases by probabilistic logic sampling of the original Alarm belief network. Herskovits indicates that the original network had 37 nodes, 46 arcs, and 10^{17} elements in its joint probability distribution.

When [Cooper and Herskovits, 1992], [Herskovits, 1991] used Kutato to reconstruct the Alarm database (Figure 5.7), they achieved a structure almost identical to the original, except for the removal of one arc (which actually yielded a more efficient structure). For my tests of METD2, I was not interested in recreating the connections between *observable* variables, but between observable and *hidden* variables. Thus, when I used the Alarm structure to create data, I filtered the experimental results to only consider subsets of observable variables. I was interested in finding, for a given subset of observable variables, how accurately METD2 could reconstruct the hidden variables. From the previous experiments with the data-generators described in Section 4.1, we saw that the algorithm handled non-tree based graphical structures by finding the closest tree approximation.

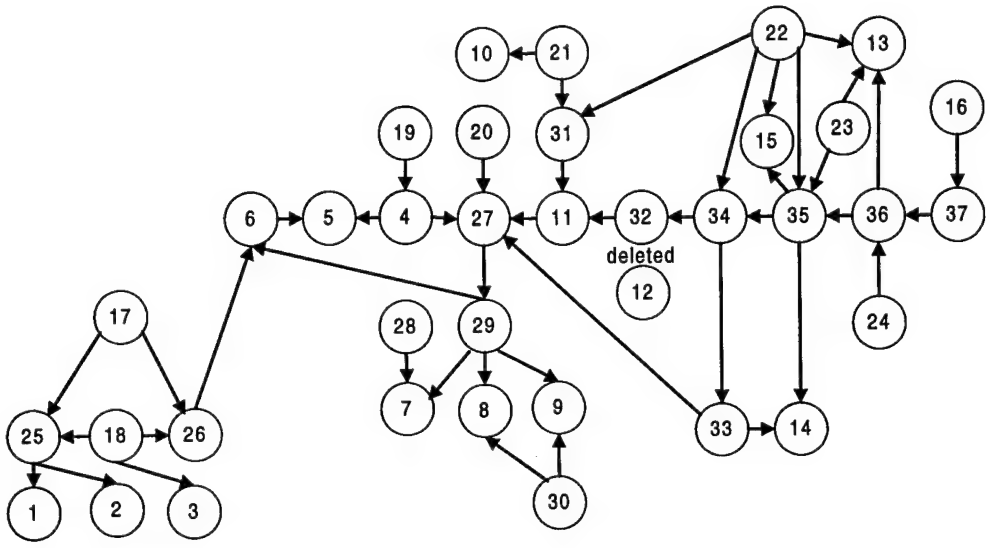


Figure 5.7 Alarm Database (Showing Arc Deleted by K2 Algorithm).

Also unlike the K2 and Kutato approach, METD2 does not determine the likelihood of each possible arc connection between nodes and then choose the connection which is most likely. Instead, it assesses the correlation error between connection, and chooses a connection which will minimize the correlation error.

5.2.2 Relationships Found on the Alarm Database

As expected, the results obtained from running METD2 on ALARM were considerably different from the results obtained by [Cooper and Herskovits, 1992]. Recall the differences between the two systems. For the first part, METD2 was not designed to find relationships among the observable variables like K2. Secondly, METD2 forces the data to fit to a tree-structure, whereas K2 had no preference for any particular structure. While K2 was striving to maximize the likelihood of the resulting graphical structure on each of its iterations, METD2 on each of its

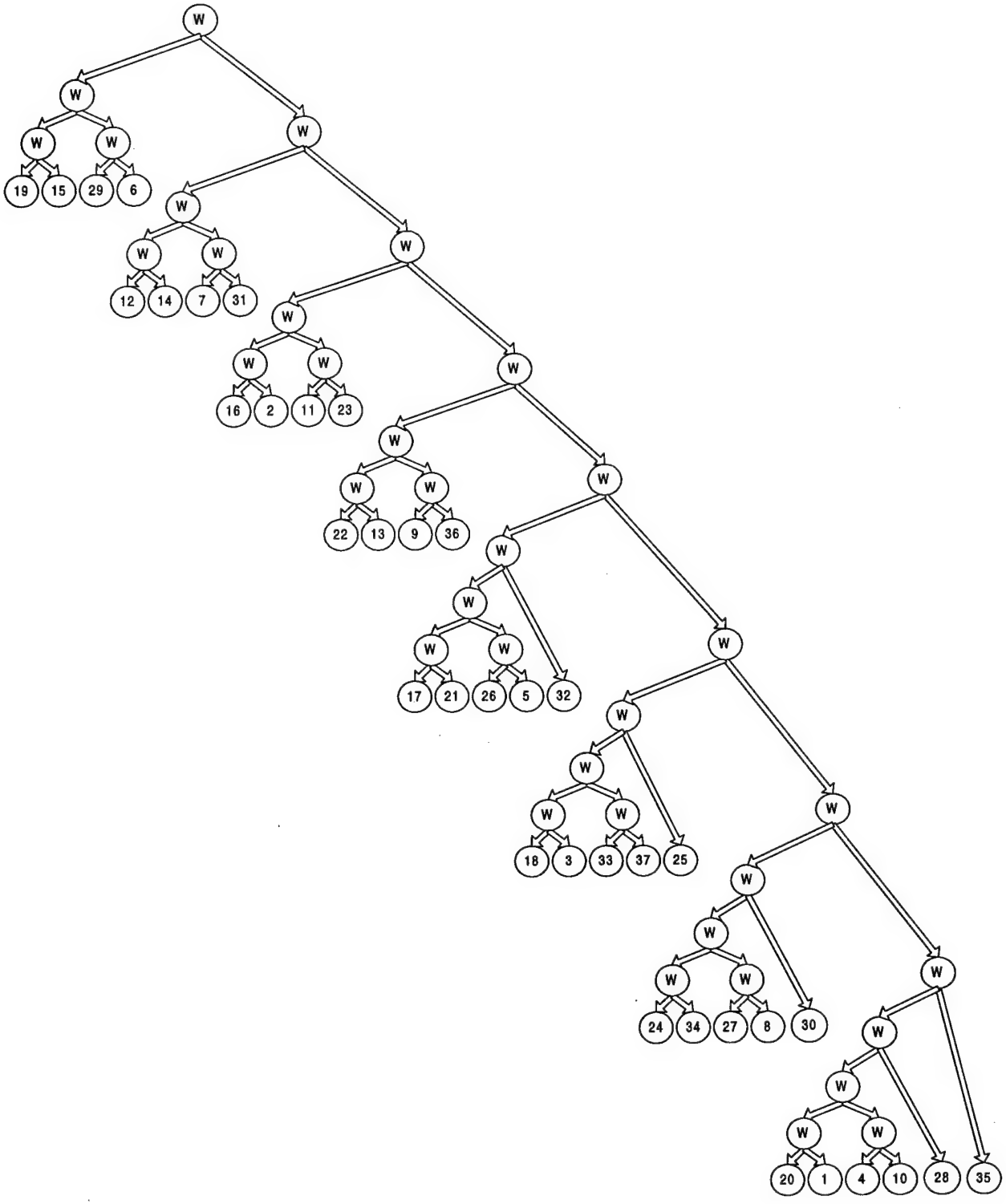


Figure 5.8 METD2 Constructed Belief Network for the Alarm Database.

iterations was striving to minimize the cross-correlation error of all structures created and joined. In short, K2 was designed to find structure in the presence of complete data, while METD2 was designed to find structure in the presence of missing data.

Examining METD2's hidden-valued structure for Alarm and comparing it to the structure generated by K2 reveals several interesting features. To start with, METD2 seems to generate three kinds of associations, if we use the original Alarm network as a base-line. The first type of association, is what I call an *inverted hidden-valued relationship*, such as the join in the lower right corner of Figure 5.8 of node 28 with node 35. Referring to Figure 5.7, we see that nodes 28 and 35 both cause node 7. While node 28 causes node 7 directly, node 35 has a causality chain of five other nodes between it and node 7. If we collapse this causality chain so that 35 directly causes node 7, and reverse the direction of the arcs, then we have a hidden-valued relationship.

The second type of association that METD2 seems to generate in Figure 5.8 if we use the original network of Figure 5.7 as a baseline is to group direct and indirect causation. For example, the sub-tree in the lower right-hand corner of Figure 5.8 consisting of nodes 24, 34, 27, 8, and 30, is a grouping of nodes which directly and indirectly cause each other. Node 24 causes node 34 with two intervening nodes. Node 24 causes node 8 with seven intervening nodes. Node 34 causes node 27 with three intervening nodes. Node 27 causes node 8 with one intervening node. And node 30 causes node 8 directly. Thus these five nodes should be strongly correlated sense they either directly or indirectly cause one another.

The third type of association evident from METD2's hidden-valued structure of Figure 5.8 are *new/unknown relationships*, such as the join in the lower right hand corner of Figure 5.8

of nodes 20 and 1 or of nodes 4 and 10. Finding these nodes in the original Alarm structure of Figure 5.7, we see that 20 only bears an oblique causal relationship to nodes 1, and that likewise, node 4 only bears an oblique causal relationship to node 10. Neither pair of nodes directly or indirectly cause each other. Their only interaction in the original Alarm database seems to be from the parents of nodes to other nodes. Thus, this mysterious association found by the algorithm calls for later examination.

Overall, the results obtained by METD2 on the Alarm database (Figure 5.8) seem to reflect some of the same relationships in the original domain model (Figure 5.7). By joining nodes which indirectly cause, or get caused by, other nodes, the METD2 algorithm creates a more "compact" representation of the domain—one showing relationships which are both obvious and obscure. Once more, this area would benefit from further research.

5.3 Asymptotic Convergence of METD2

For noisy data, the METD2 algorithm shows great sensitivity to the correlation coefficients, as expected. Increasing the number of cases in the database leads to increasingly more accurate results. This confirms results attained by [Henrion et al, 1996]. They show the robustness of medical diagnostic Bayesian networks when the networks are exposed to noisy data. They found that the structure of the Bayesian network encodes a "great deal of information" and that the individual probabilities are less important than the structure, as long as the probabilities are "in the right direction." Even after two standard deviations of noise added to the data, they found that the networks could still gave the correct diagnosis the majority of the time.

Nevertheless, both further theoretical research to prove asymptotic soundness and further empirical research to verify theoretical proofs would contribute much to METD2. Figure 5.9 shows the trees constructed by applying METD2 to different sample sizes of noisy data drawn from the artificial domains illustrated in Figures 5.1-5.3. In each row, nodes combined in the graphical structure are grouped together in parenthesis. The outermost parenthesis, which designate the complete, rooted tree, are not shown. The last row, where $N = 10{,}000$ corresponds to the "asymptotic" behavior of the algorithm. Each of the trees listed in the last row have graphical illustrations in Figures 5.4-5.6, respectively.

N	Balanced-Binary Tree	Unbalanced-Binary Tree	Unbalanced -Trinary Tree
100	(0 3 5 1) (2 4 6 7)	((1 2 3 4) 0 5) 6	((1 3 4 6) 5) (0 2)
1,000	(0 1 4 6) (2 3 5 7)	((0 1 3 4) 2) (5 6)	((2 4 5 3) (1 6)) 0
10,000	((1 4 5 6) 2 3) (0 7)	((1 2 4 5) 05) 3	(((1 2 3 6) 5) 4) 0

Figure 5.9 Trees Constructed on *N* Noisy Samples Drawn from Artificial Databases.

Theoretically, any Bayesian belief network can be reduced to a structure with a single node, where the single node contains the entire conditional probability table and joint probability tables for the problem domain. Established approaches to designing belief models attempt to capsulate a human expert's knowledge by reproducing its semantic associations in graphical form, and then assigning some numerical meaning to that graph. Yet, in the abstract sense, all the concepts represented by nodes in the domain model could be reduced to a single concept—the whole problem. Thus, what the results of Figure 5.9 seem to indicate is that the hidden structure for a domain could be collapsed into a single node as well.

5.4 Memory, Storage, and Time

The first potential difficulty occurs with databases which contain large numbers of variables (e.g. > 100), because the computational complexity of METD2 is $O(N^5)$. If n = 100, the algorithm with perform some multiple of 100 billion calculations. To see how quickly the number of variables can become a problem, consider an implementation of the combinatorial function in the language C++ on a UNIX operating system. The maximum size integer permitted is 0x7FFFFFFF which is 2,147,483,647 base 10. $C(500,4) = 2.5 \times 10^9$, which exceeds this number.

The typical run-time behavior of the algorithm reflects the inductive bias when applying the cross-correlation error—since the cross-correlation between a tree and another tree or leaf or leaf pair is the worst of all possible combinations, the algorithm tends to select leaf quadruples before other objects. The algorithm tends to build most or all of the initial quadruples first, then combine the remaining three or fewer leaves with one to three of the existing trees, and then perform tree combinations.

Figure 5.10 shows actual running times for optimized C++ code implementing the METD2 algorithm on a dedicated 50 MHz Heward Packard Series 9000/715 with 64 MB RAM. Each trial of METD2 used a database of 10,000 randomly generated cases, i.e. any correlation of the variables would be completely random. Each case had a varying number of variables. As expected, the run-time for the algorithm increased with increasing numbers of domain variables. Note, that these results are for an implementation of METD2 which was less than optimal—the dynamic structures were not optimized and much debugging code was left in the program as it

ran. Fielded applications implementing this algorithm would be optimized for the problem domain and use more efficient data structures.

Number Observable Variables	Running Time (minutes: seconds)
20	0:43
25	1:08
30	2:00
35	4:10
40	7:03
45	15:37
50	32:00

Figure 5.10 Running Times on Randomly Generated Databases Containing 10,000 Samples.

6. CONCLUSION

In this chapter, I conclude the thesis. First, I review the contents of the previous chapters. Then, I discuss some of the results described in Chapter 5 and consider the qualitative performance and effectiveness of METD2. Next, I outline the contributions made to probabilistic-network research by this thesis. Finally, I describe open issues for future research in belief networks and the problem of finding hidden structure.

6.1 Review of Thesis Contents

Chapter 1 of this thesis provided an introduction to Bayesian belief networks and the problem of learning hidden structure from data. The discussion spanned probability theory, statistics, and artificial intelligence.

Chapter 2 of this thesis provided an introduction to the METD2 algorithm, giving a high-level pseudo-code for METD2 and showing the outputs from an application of the algorithm to an example problem. The example run of METD2 required three complete passes to generate a hidden structure for a domain consisting of 11 observable variables.

Chapter 3 of this thesis discussed previous and related research to METD2. I first examined two previous Bayesian methods for learning structure from complete data (without hidden variables)—Kutato/K2 and Spanning Trees. The Spanning Tree approach did not

determine parameters, only structure, but the Kutato/K2 methodology determined both. The next section examined a non-Bayesian learning method—Artificial Neural Networks/Error-Based Back-Propagation, a method well suited for optimizing parameters given a particular graph topology. The final section considered Bayesian methods for learning hidden structure, such as AutoClass and more generalized methods. Here, I examined the direct predecessors to METD2—Exact Tree Decomposition and Minimum Error Tree Decomposition I.

Chapter 4 provides the theoretical foundations, algorithmic details, and complexity analysis for METD2. The complexity analysis showed that the worst case behavior of METD2 is the same as that of METD1, namely $O(N^5)$, while its best case analysis is one polynomial degree better—$O(N^4)$. The theoretical description assumed, as noted in Chapter 2, that a tree-dependent distribution would accurately describe the data. The detailed section on component algorithms showed how to implement the steps of METD2.

Finally, **Chapter 5**, tested METD2 on artificially-created and real-world problem domains. Results in both testing realms were surprising and clearly indicate a strong need for further testing. The results implied that METD2 finds an equivalent tree-structure where possible, or the most compact, functionally-equivalent tree-structure. Section 6.2 contains further analysis of the empirical results.

6.2 Further Discussion of Results

The METD2 algorithm does construct a Bayesian graphical structure consisting of leaf nodes which are the observable variables in some problem domain, and internal nodes which represent the hidden concepts in the problem domain. By assuming a tree-structure for the hidden

variables, METD2 will not be able to reconstruct accurately the hidden structure for an arbitrary graph. Instead, it will find the closest—as defined by minimal cross-entropy—tree-structured approximation for the graph. Note that this is not a limitation. Indeed, the alternate graphical structure may be more compact and efficient than the original, arbitrary graph. Certainly, its tree-structure insures that computation/propagation of new information will occur at least as quickly, and more quickly in the general case, than the original graph.

The algorithm finds *a* tree-structured representation of the data, but data, even without noisy correlation coefficients, may still have multiple trees that accurately fit the data. Thus, two data sets drawn from the same problem domain may yield two different network structures. What the results from Chapter 5 show is that domains tend to have non-unique tree-dependent hidden-structures, and that multiple unique hidden structure exist for the data. The example run from Chapter 2, showed that for some domains with exact data, there is no unique representation for the underlying domain model. What I expect the algorithm to do in the asymptotic case, i.e., as the number of data samples approaches the population size, is to reproduce the *most* exact tree-structure for that data. An open issue is whether or not, domains which do not have an underlying tree structure, have a unique tree-structured approximation.

One way to speed-up the algorithm is to perform only tree-to-tree combinations until some threshold has been crossed, and then to consider other types of combinations. Another option, is to ignore the lower $n\%$ of the sorted quadruples since they are likely never to be used. However, I believe that the general key to reducing the running time will be finding a was not to

generate all the N^4 quadruples in the first place. Further research should determine if this is feasible, and if so, how to implement it.

6.3 Contributions of This Thesis

This thesis showed that *Minimum-Error Tree Decomposition II*, METD2, can graphically decompose a joint distribution by assuming a tree-dependent distribution underlies the domain model. METD2 uses the tree-dependency assumption to marginalize a joint distribution of n variables into a distribution on $n+1$ variables and one hidden variable, w, such that the n observable variables are conditionally independent of each other given the additional hidden variable w, and w forms a binary tree whose leaves are the n variables in the source database and whose $n+1$ interior nodes are the components of the hidden variable w. No other researchers into the problem of finding hidden variables or finding structure for missing data since Liu and Pearl have considered a tree-structured domain assumption. They have either assumed a single hidden variable causes all observable variables or that n hidden variables cause the observable variables.

The Tree Decomposition method was first proposed by [Pearl, 1988] and [Liu, 1990]. METD2 extends the approach proposed by [Liu, 1990] of METD1. METD2 is the first implementation of such an approach that fits the observed data behavior to a tree-dependent distribution. This thesis adds a new tool—the minimum cross-correlation error metric—to the set of tools for computer-based construction of probabilistic domain models. METD2 provides a non-parametric, domain-independent, computationally-efficient system for finding hidden structure.

This thesis provides the first experimental results of Minimum-Error Tree Decomposition. In Chapter 5, I compared METD2's results on the ALARM database to those obtained by another structuring algorithm, K2. METD2 found, in general, different relationships among the data than the K2 approach—METD2 found relationships in the database that were not discovered by K2. This thesis provides the first detailed examination of the actual run-time behavior of the METD1/2 approaches. It also provides the first complexity analysis and calculation of the asymptotic bounds for the METD2 approach. Although the worst-case time complexity for METD1/2 is $O(N^5)$, the typical run-time behavior is $O(N^4)$.

6.4 Open Issues for Future Research

This research was conducted under time and scope constraints which prevented consideration of a host of issues, some of which are critical for using Tree Decomposition in end-user problem domains. In addition, the research results from Chapter 5 raise new questions. Also, the field of finding hidden belief networks, in general, has many open areas for research.

6.4.1 Meaning of Hidden Variables

[Pearl, 1988; 383] has described the role played by hidden or abstract variable in human thinking as simplifying the reasoning task. He argues that human beings use hidden variables to decompose complex problems. For example, in medicine doctors group together symptoms which tend to occur at the same time and call them "maladies" or "syndromes" or "complexes." The introduction of the "hidden-variable" decomposes the set of symptoms which tend to occur at the same time and permits the medical professional to conceptualize them as being

conditionally independent of each other once he knows their "hidden-cause." For example, a doctor groups several symptoms which tend to occur together and calls them a malady or a syndrome. Knowing the malady or syndrome makes the several symptoms conditionally independent of each other. Hence, one possible area for further research is to determine what, if any, correspondence lies between the "concepts" created by METD2 and real-world concepts. Like the application of AutoClass to NASA's spectrometry database, this process could lead to scientific discovery.

6.4.2 Combination with Other Methods

Combination with Complete-Data Structuring Algorithms. Dedicated methods for finding associations among the observable variables in a database, i.e. for complete data, return one set of relationships, while methods for finding associations between the observable variables and constructed hidden variables form another. Combining these two approaches into a hybrid approach should yield superior domain models—models which are more compact, have a lower degree, and therefore, are easier to compute and generate more accurate models.

Combination with Generalized Search Methods. All of the more general methods for finding hidden structure (described in Section 3.3) construct a domain model by adding/removing/inverting a single arc at a time, while METD2 adds up to 4 arcs per iteration. Thus, METD2 tends to generate larger hidden-valued models than some other methods, but it generates them using less computational time. Creating a hybrid approach with a more general method, such as the BIC/MDL criterion [Draper, 1993], for example, should yield more compact, yet more expressive, domain models.

Combination with ANN Methods. This thesis did not implement the parameter-estimation portion of the METD1 algorithm proposed by [Liu et al, 1990]. One alternative approach for determining the parameters is neural network-based optimization. The strength of the Bayesian method is its firm mathematical grounding and its strong connection to symbolic reasoning, while the strength of the artificial neural network (ANN) approach is its strong numerical or sub-symbolic connection which permits it to approximate any function. Combining the two approaches should yield systems with both domain-specific graphical structures and domain-specific parameters. Since Bayesian methods are superior at learning structure, and ANN methodologies at learning parameters, combining the two approaches should lead to superior systems.

There are already some results from hybrid Bayesian-belief network/Artificial neural network approaches. For example, [Neal, 1992] indicates some success through such a hybrid approach. [Buntine, 1996] has a good survey on the literature of combining ANN and BBN methodologies and conversion from one form to another. METD2 performs a batch-processing of the data, rather than creating the hidden-valued belief-network incrementally. Using the ANN paradigm is one possible tactic to adaptive (iterative) learning from additional to supplement or even replace the batch mode examined in this paper.

6.4.3 Generalization

Generalizing the Cross-Correlation Criterion. To determine the theoretical asymptotic limit of the structures built by METD2 requires a more general equation for the minimum cross-correlation error (Equations 6-10). A proof of correctness is important for further research into

the correlation-error/Tree Decomposition approach. The combination operations described in Chapter 4 may be incorrect, even though the limited empirical evidence in Chapter 5 implies that they are not, and may not generate increasingly more likely tree-structures given increasingly larger numbers of cases in a database. In addition, a more general metric will facilitate combination with other methods.

Generalization to Rooted n-ary Trees. Rooted trinary trees, or even rooted trees of a higher degree, do not violate the d-separation and other conditional independence constraints for finding a tree-dependent distribution. Having higher fan-in at each node should yield more sophisticated, but probably more complicated, models. The best approach is to permit variable fan-in at each node, such that any node in the general tree structure is a b-tree, with a unique parent but any number of children. This approach is the next logical step in generalizing the METD1/2 approaches and should yield a more general algorithm than METD2.

Generalization to Polytrees. The METD2 approach assumes a tree-structured model, but a heuristic-based greedy search similar to METD2 might use a different assumed model than a binary tree, such as a polytree. The next logical step after generalizing the METD2 approach from binary to b-trees is to generalize it further from b-trees to polytrees. The advantages of assuming a tree as the underlying domain model were detailed in Chapter 1. In the case of a polytree, as opposed to a binary tree, distribution assumption, the potential advantages would be to create a method with greater generality than METD2. However, [Pearl, 1988; 407] notes at least one major difficulty with attempting to attend the tree structure to a polytree: determining

the parameters associated with distributions that are not binary or normal is much more complicated and rarely leads to a unique solution.

6.4.4 Other Modifications

Completing the Parameter-Determination Algorithm. This thesis did not implement parameter determination. Yet, further scientific research and/or commercial application would benefit from METD2 actually generating the conditional probabilities and the prior probabilities for the hidden variables added to the network structure. The conditional and prior probabilities on the hidden nodes are necessary for data fusion and propagating new information throughout the network. Also, having the conditional and prior probabilities on the hidden nodes should yield the likelihood for the entire network, which is essential for a quantitative comparison with other systems, such as K2.

Changing Arc Directions. METD2 assumes that the hidden variables *causes* the observable variables, which yields one, unique arc assignment for each arc: *from* the hidden variables *to* the observed variables. Permitting arbitrary arc directions may generate more accurate and more compact domain models.

Changing the Inductive Bias. The inductive bias implemented in METD2 heavily weighted the algorithm toward choosing quadruples, first, when building the forest of trees, and then, toward combining into a heavily unbalanced tree. Relaxing or changing the bias will lead to significant tree structures.

Scaling and Parallelizing. From Section 5.3, where I considered memory, storage, and time, we see that scalability is an important area for further research. Given the large level of

many, identical, independent operations occurring, the algorithm is clearly parallelizable. Parallelizing the algorithm will make it scaleable to problem domains consisting of hundreds of variables.

REFERENCES

[Aliferis and Cooper, 1994] Aliferis C. and Cooper G. "An evaluation of an algorithm for inductive learning of Bayesian belief networks using simulated data sets. In Proceedings of Tenth Conference on Uncertainty in Artificial Intelligence. Seattle, WA: Morgan Kaufmann, 1994. pp 8-14.

[Buntine, 1966] Buntine W.. "Real World Application of Learning Methods." Talk slides: 1996 UAI Tutorial 7/31/96.

[Charniak, 1991] Charniak E. "Bayesian Networks without Tears." AI Magazine, Winter 1991.

[Cheesman et al, 1989] Cheesman P., Kelly J., Self M., Stutz J., Taylor W., Freeman D. "AutoClass: A Bayesian Classification System." Proceedings of the Tenth International Joint Conference on Artificial Intelligence. pp 264-270. 1989.

[Chickering, 1996] Chickering D. "Learning Equivalence Classes of Bayesian Network Structures." Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence. Portland, OR: 1-4 Aug. 1996. pp 150-157.

[Chickering and Heckerman, 1996] Chickering D. and Heckerman D. "Efficient Approximations for the Marginal Likelihood of Incomplete Data Given a Bayesian Network." Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence. Portland, OR: 1-4 Aug. 1996. pp 158-168.

[Chow and Liu, 1968] Chow C. and Liu C. 1968. "Approximating Discrete Probability Distributions with Dependence Trees." IEEE Transactions on Information Theory. IT-19:369-71.

[Cooper, 1996] Cooper G. "A Constraint-Based Approach to Learning Probabilistic Networks from Observational Data." Talk slides: UAI Tutorial 7/31/96.

[Cooper and Herskovits, 1992] Cooper G. and Edward Herskovits. "A Bayesian Method for the induction of Probabilistic Networks form Data." Machine Learning, vol. 9, no. 4, Oct. 1992.

[Draper, 1993] Draper D. "Assessment and Propagation of Model Uncertainty." TR 124, Dept. of Statistics, University of California, Los Angeles. LA: 1993.

[Friedman and Goldszmidt, 1996] Friedman N. and Goldszmidt M. "Learning Bayesian Networks with Local Structure." Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence. Portland, OR: 1-4 Aug. 1996.

[Hassoun, 1995] Hassoun, M. Fundamentals of Artificial Neural Networks. Cambridge: The MIT Press, 1996. Chap: 1.1, 3.1., 2.2, 2.3.3.

[Heckerman, 1995] Heckerman D. "A Tutorial on Learning with Bayesian Networks." Microsoft Research: Advanced Technology Division. Technical Report MSR-TR-95-06. ftp://ftp.research.microsoft.com/pub/dtg/david/tutorial.ps.

[Heckerman et al., 1995] Heckerman D., Geiger D., and Chickering D. "Learning Bayesian Networks: The Combination of Knowledge and Statistical Data." Machine Learning. 20:197-243.

[Henrion et al, 1996] Henrion M., Pradhan M., Del Favero B., Huang D. P., Gregory, and O'Rorke P. "Why Is Diagnosis Using Belief Networks Insensitive to Imprecision in Probabilities?" Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference. San Francisco: Morgan Kaufmann Publishers, August 1-4, 1996. pp. 307-314.

[Herskovits, 1991] Herskovits E. "Computer-Based Probabilistic-Network Construction." Department of Computer Science and Medicine, Stanford University. Report No. STAN-CS-91-1367 *Thesis*. June 1991. pp 1-38, 109-129, 160, 185-191.

[Hsu, 1995] Hsu, William H. "Probabilistic Reasoning for Knowledge-Based Systems: An Overview of Belief Networks and Knowledge-Based Simulation." Talk slides: KBS Group, UI, 10/03/95.

[Laskey and Martignon, 1996] Laskey K. B. and Martignon L. "Bayesian Learning of Loglinear Models for Neural Connectivity." Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence. Portland, OR: 1-4 Aug. 1996.

[Liu et al, 1990] Liu L., Wilkins D. C., Ying X., and Bian Z. "Minimum Error Tree Decomposition." Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence. Cambridge, MA: 27-29 July 1990.

[Neal, 1990] Neal, R. M. "Learning stochastic Feedforward Networks." CRG-TR-90-7. Department of Computer Science, University of Toronto. 1990. Http:// www.cs.utoronto.ca/~radford/belief-net.abstract.html.

[Madigan and Raftery, 1994] Madigan D. and Raferty A. "Model Selection and Accounting for Model Uncertainty in Graphical Models Using Occam's Window." Journal of the American Statistical Association. 1994. 89:1535-1546.

[Neopolitan, 1990] Neopolitan R. E. Probabilistic Reasoning in Expert Systems: Theory and Algorithms. New York: Jon Wiley & Sons, Inc., 1990.

[Pearl, 1988] Pearl J. "Learning Structure from Data," Chap. 8, Probabilistic Reasoning in Intelligent Systems, San Francisco: Morgan Kaufmann Publishers, Inc., 1988. pp. 381-414.

[Pearl, 1990] Pearl J. "Fusion, Propagation, and Structuring in Belief Networks." North-Holland: Elsevier Science Publishers B.V., 1986. Shafer and Pearl. Ed. Readings in Uncertainty Reasoning. San Mateo: Morgan Kaufmann Publishers, 1990.

[Rumelhart, 1986] Rumelhart D. E. "Learning by Internal Representation by Error Propagation, " in Rumelhart (ed.), Parallel Distributed Processing Explorations in the Microstructure of Cognition, Vol. 1: Foundations, Cambridge, Mass.: The MIT Press, 1986.

[Spirtes and Meek, 1995] Spirtes P. and Meek C. "Learning Bayesian Networks with Discrete Variables from Data." In Proceedings from First International Conference on Knowledge Discovery and Data Mining. Montreal, QU: Morgan Kaufmann, 1995.